CS490: Problem Solving in Computer Science

Lecture 3: Input/Output

Dustin Tseng
Mike Li

Wednesday January 4, 2006

Outline

- Input in Java
- Output in Java
- I/O in C++

- Input in Java
- Output in Java
- I/O in C++

Before Java 1.5

Before java 1.5, usually we read input line by line and the rest is string processing.

```
// To read from standard in:
BufferedReader cin = new BufferedReader(new InputStreamReader(System.in));
// To read from file:
BufferedReader fin = new BufferedReader(new FileReader("myfile"));
// To read from string:
BufferedReader sin = new BufferedReader(new StringReader("read this string"));
```

After reading in the lines, we parse it. We will use standard input for example.

Before Java 1.5

```
String line;
while( (line = cin.readLine()) != null) { // this checks if there are more input

    // to get rid of leading or trailing whitespaces
    // quite useful, in case some extra space was in the file by accident
    line = line.trim();

    // suppose we have an integer
    int myInt0 = Integer.parseInt(line);

    // suppose we have a binary integer
    int myInt1 = Integer.parseInt(line, 2);

    // suppose we have a double, and so on....
    double myDbl = Double.parseDouble(line);

    // suppose we have a whole bunch of space delimited integers
    // e.g. "1 4 2 3 5"
    String[] toks = line.split(" +"); // " +" is a regular expression
    int[] myInts = new int[toks.length];
    for(int i = 0; i < toks.length; i++)
        myInts[i] = Integer.parseInt(toks[i]);

    // read up the API of split to see its behavior and options
    // when there are trailing, leading, multiple delimiters
}
```

With Java 1.5

Now, with Java 1.5, we use `Scanner`. `Scanner` automatically deals with whitespaces (similar to `cin` in C++)

```
Scanner in;
// here is scanner from standard in, string, and file
in = new Scanner(System.in);
in = new Scanner("scan this string");
in = new Scanner(new File("myFile"));

// you can change the deliminters to something other than
// whitespace by passing in a second argument

// e.g. read a bunch of ints
while(in.hasNextInt()) {
int garbage = in.nextInt();
}

// similarily, there is nextDouble(), nextBigInteger(), etc.
```

With Java 1.5

Sometimes we still need to do line by line processing.
e.g. when you want to sum up (space delimited) integers in a line, but you don't know how many integers are in the line

```
String lineOfInt = in.nextLine();
String[] moreToks = lineOfInt.trim().split(" +");
int sum = 0;
for(String s : moreToks) sum += Integer.parseInt(s);
```

For even more nasty things, check out details of `split`, as well as the `String` API

With Java 1.5

Warning: Take extreme care when switching between `nextInt()` and `nextLine()`. e.g. If input is:

```
1
1 2 3 4 5
```

After `readInt()`

```
1
 ^
1 2 3 4 5
```

The next `readLine()` gives an empty line, and moves the caret to next line

```
1
1 2 3 4 5
^
```

Now, the second `readLine()` will give you the meaningful stuff.

- Input in Java
- Output in Java
- I/O in C++

Java Output

Java output is pretty straight forward. e.g. if we want to print the following:

```
int a = 1;
char b = 'Z';
String c = "ho ho ho!";
```

We write:

```
System.out.println("" + a + b + c);

\\ or you can do it one by one
System.out.print(a);
System.out.print(b);
System.out.println(c);
```

To print to a file use:

```
PrintWriter fout = new PrintWriter(new File("outputFile"));
```

Example: Interger Array

Print the contents of a integer array seperated by spaces, in base 3

```
int[] myArray = new int[]{1, 4, 2, 3, 4, 2};
for(int i = 0; i < myArray.length; i++) {
    if (i != 0) System.out.print(' ');
        System.out.print(Integer.toString(myArray[i], 3));
}
System.out.println();
```

Example: Floating Points

Printing a floating point number is more work. Usually there is a certain format (e.g. number of decimal places). Thanks to Java 1.5, we now have `format` (similar to C/C++ `printf`)
Suppose we want to print the following integers and doubles

```
int[] ints = new int[]{3, 2, 15};
double[] doubles = new double[]{2.2, 0.43, 25.267};
```

into:

```
 3  2.20
 2  0.43
15 25.27
```

do:

```
for(int i = 0; i < 3; i++) {
    System.out.format("%2d %5.2f\n", ints[i], doubles[i]);
}
```

More Formatter

There's a bunch of other options. e.g.

- `-` means left justify
- `0` means leading zeros

`%s` is for strings

```
for(int i = 0; i < 3; i++) {
    System.out.format("%02d %-5.2f\n", ints[i], doubles[i]);
}
```

gives:

```
03 2.20
02 0.43
15 25.27
```

Even More Formatter

Finally, in the spirit of variable arguments

```
String[][] words= new String[][]{
    {"First", "Last", "Fav Food", "e-mail"},
    {"Winnie", "Pooh", "Honey", "pooh@100acrewoods.net"}
};
for(String[] sa: words)
    System.out.format("%4$-25s %1$-10s %2$-10s\n", sa);
```

gives:

```
e-mail                    First      Last
pooh@100acrewoods.net     Winnie     Pooh
```

For a complete description, read Java API:

`http://java.sun.com/j2se/1.5.0/docs/api/java/util/Formatter.html`

- Input in Java
- Output in Java
- I/O in C++

Overview

I/O in C++ generally involves the following standard libraries that deals with `stream` class:

- `iostream`: Standard I/O
- `fstream`: File I/O
- `sstream`: Convert `string` to `stream`
- `iomanip`: I/O manipulation
- `cstdio`: I/O functions inheretied from C

To use these libraries, use `#include`. e.g.

```
#include <iostream>
\\ we also need to specify the namespace by:
using namespace std;
```

iostream

To parse input from standard input stream, we simply use `cin` with the extraction operator >>, which automatically loads all standard types. e.g.

```
int n;
double f;
string s;
cin >> n >> f >> s;
```

This works similarly to `Scanner` in Java. Both of them discard whitespaces (`'\n'`, `'\t'`, `' '`, `etc`) between inputs. So

```
7 3.14 pie
```

and

```
7
3.14          pie
```

will produce the same `n`, `f`, `s`.

iostream

Similarly, we have `cout` and `<<` to deal with standard output. e.g.

```
cout << n << f << s;
```

would produce

```
73.14pie
```

To add space or end of line, we can write:

```
cout << n << " " << f << endl << s;
```

which would produce

```
7 3.14
pie
```

fstream

File I/O is really simple in C++. First we need specify the input and output stream. The rest would then be the same as standard I/O. e.g.

```
ifstream fin("input.txt");
fin >> n >> f >> s;

ofstream fout("output.txt");
fout << "case " << n << ":\n" << s << "= " << f << endl;
```

This is what output.txt will look like:

```
case 7:
pi= 3.14
```

Checking I/O errors would increase robustness of the program, but generally not a concern in problem solving.

sstream

Sometimes we would like to parse input by line rather than by value. One case mentioned before was to add all numbers in a line. To do this in C++, we can write:

```
string line;
int n=0;

while(getline(cin, line, '\n')) {
    int sum=0, i;
    stringstream strin(line);
    while(strin >> i)
        sum += i;
    cout << "line " << ++n << ": " << sum << endl;
}
```

Note: similar to `NextLine()`, we also need to be careful with trailing `'\n'` while using `getline()`. Usually we solve this problem by calling `getline()` again on a dummy string variable.

iomanip

Output stream can be easily manipulated by using `iomanip`:

```
\\ set base for integers (8, 10 or 16)
cout << setbase(16) << 100 << endl;
\\ hex, dec and oct are predefined, so equivalently:
cout << hex << 100 << endl;

\\ set precision for floats
cout << setprecision(1) << 3.14 << endl;
cout << setprecision(3) << 3.14 << endl;

\\ set whitespace
cout << setw(3) << 7 << endl;

\\ set fill characters
cout << setfill('0') << setw(3) << 7 << endl;
```

The output would be:

```
64
64
3.1
3.14
  7
007
```

cstdio

`iomanip` often does not offer enough formatting functionalities, we usually turn to the more powerful `printf` from C.
Let's use the same example used for Java and suppose we want to print

```
int ints[3] = {3, 2, 15};
double doubles[3] = {2.2, 0.43, 25.267};
```

into:

```
 3  2.20
 2  0.43
15 25.27
```

In C++ we can write:

```
for(int i = 0; i < 3; i++) {
    printf("%2d %5.2f\n", ints[i], doubles[i]);
}
```

For a good reference, visit:

`http://www.cplusplus.com/ref/cstdio/printf.html`

Two More

`cctype` :

- `tolower()`, `toupper()`
- Convert character cases

`climits` :

- `INT_MAX`, `INT_MIN`, `UINT_MAX`, `ULONG_MAX`
- Useful constants

What Else?

- Content in terms of amount/speed
- More exciting stuff coming up
- Online judge
- Topic preference
- Order of the presentations