ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στον Προγραμματισμό
Introduction to Programming

Διάλεξη 10: Μορφοποίηση Εισόδου/Εξόδου

Γ. Παπαγιαννάκης

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

HY-150 Προγραμματισμός
CS-150 Programming

Lecture 10: Customizing I/O

G. Papagiannakis

Overview

- Input and output
- Numeric output
  - Integer
  - Floating point
- File modes
  - Binary I/O
  - Positioning
- String streams
- Line-oriented input
  - Character input
  - Character classification

Kinds of I/O

- Individual values
  - See Chapter 4, 10
- Streams
  - See Chapters 10-11
- Graphics and GUI
  - See Chapters 12-16
- Text
  - Type driven, formatted
  - Line oriented
  - Individual characters
- Numeric
  - Integer
  - Floating point
  - User-defined types

Type	Size in bits	Format	Value range	
			Approximate	Exact
character	8	signed (one's complement)	-127 to 127	
		signed (two's complement)	-128 to 127	
		unsigned	0 to 255	
integral	16	signed (one's complement)	$\pm 3.27 \cdot 10^4$	-32767 to 32767
		signed (two's complement)		-32768 to 32767
		unsigned	0 to $6.55 \cdot 10^4$	0 to 65535
	32	signed (one's complement)	$\pm 2.14 \cdot 10^9$	-2,147,483,647 to 2,147,483,647
		signed (two's complement)		-2,147,483,648 to 2,147,483,647
		unsigned	0 to $4.29 \cdot 10^9$	0 to 4,294,967,295
	64	signed (one's complement)	$\pm 9.22 \cdot 10^{18}$	-9,223,372,036,854,775,807 to 9,223,372,036,854,775,807
		signed (two's complement)		-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
		unsigned	0 to $1.84 \cdot 10^{19}$	0 to 18,446,744,073,709,551,615
floating point	32	IEEE-754	$\pm 3.4 \cdot 10^{\pm 38}$ (~7 digits)	• min subnormal: $\pm$ 1.401,298,4 $\cdot 10^{-47}$ • min normal: $\pm$ 1.175,494,3 $\cdot 10^{-38}$ • max: $\pm$ 3.402,823,4 $\cdot 10^{38}$
	64	IEEE-754	$\pm 1.7 \cdot 10^{\pm 308}$ (~15 digits)	• min subnormal: $\pm$ 4.940,656,458,412 $\cdot 10^{-324}$ • min normal: $\pm$ 2.225,073,858,507,201,4 $\cdot 10^{-308}$ • max: $\pm$ 1.797,693,134,862,315,7 $\cdot 10^{308}$

Observation

- As programmers we prefer regularity and simplicity
  - But, our job is to meet people's expectations
- People are very fussy/particular/picky about the way their output looks
  - They often have good reasons to be
  - Convention/tradition rules
    - What does 123,456 mean?
    - What does (123) mean?
  - The world (of output formats) is weirder than you could possibly imagine

Output formats

- Integer values
  - **1234** (decimal)
  - **2322** (octal)
  - **4d2** (hexadecimal)
- Floating point values
  - **1234.57** (general)
  - **1.2345678e+03** (scientific)
  - **1234.567890** (fixed)
- Precision

(for floating-point values)
  - **1234.57** (precision 6)
  - **1234.6** (precision 5)
- Fields
  - **|12|** (default for **|** followed by **12** followed by **|**)
  - **| 12|** (**12** in a field of 4 characters)

Numerical Base Output

- You can change "base"
  - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// simple test:
        cout << dec << 1234 << "\t(decimal)\n"
                << hex << 1234 << "\t(hexadecimal)\n"
                << oct << 1234 << "\t(octal)\n";
// The '\t' character is "tab" (short for "tabulation character")

// results:
    1234        (decimal)
    4d2         (hexadecimal)
    2322        (octal)
```

“Sticky” Manipulators

- You can change “base”
  - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// simple test:
cout << 1234 << '\t'
             << hex << 1234 << '\t'
             << oct << 1234 << '\n';
cout << 1234 << '\n';        // the octal base is still in effect

// results:
1234    4d2    2322
2322
```

Other Manipulators

- You can change "base"
  - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// simple test:
    cout << 1234 << '\t'
              << hex << 1234 << '\t'
              << oct << 1234 << endl;          // '\n'
    cout << showbase << dec;     // show bases
    cout << 1234 << '\t'
              << hex << 1234 << '\t'
              << oct << 1234 << '\n';
// results:
    1234        4d2      2322
    1234        0x4d2    02322
```

Floating-point Manipulators

- You can change floating-point output format
  - general – **iostream** chooses best format using **n** digits (this is the default)
  - **scientific** – one digit before the decimal point plus exponent; **n** digits after **.**
  - **fixed** – no exponent; **n** digits after the decimal point

```
// simple test:
cout << 1234.56789 << "\t\t(general)\n"//  \t\t  to line up columns
        << fixed << 1234.56789 << "\t(fixed)\n"
        << scientific << 1234.56789 << "\t(scientific)\n";

// results:
1234.57            (general)
1234.567890        (fixed)
1.234568e+003      (scientific)
```

Precision Manipulator

- Precision (the default is 6)
  - general – precision is the number of digits
    - Note: the **general** manipulator is not standard, just in std_lib_facilities.h
  - **scientific** – precision is the number of digits after the . (dot)
  - **fixed** – precision is the number of digits after the . (dot)

// example:

```
cout << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
        << scientific << 1234.56789 << '\n';
cout << general << setprecision(5)
        << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
        << scientific << 1234.56789 << '\n';
cout << general << setprecision(8)
        << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
        << scientific << 1234.56789 << '\n';
```

// results (note the rounding):

```
1234.57       1234.567890      1.234568e+003
1234.6        1234.56789       1.23457e+003
1234.5679     1234.56789000    1.23456789e+003
```

Output field width

A width is the number of characters to be used for the next output operation

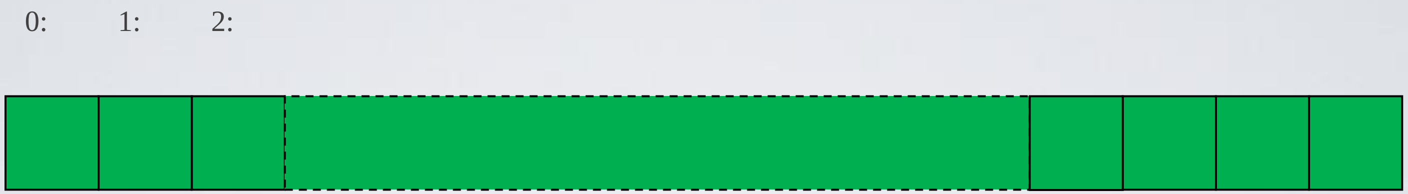
- Beware: width applies to next output only (it doesn't "stick" like precision, base, and floating-point format)
- Beware: output is never truncated to fit into field
  - (better a bad format than a bad value)

```
// example:
    cout << 123456 <<'|'<< setw(4) << 123456 << '|'
            << setw(8) << 123456 << '|' << 123456 << "|\n";
    cout << 1234.56 <<'|'<< setw(4) << 1234.56 << '|'
            << setw(8) << 1234.56 << '|' << 1234.56 << "|\n";
    cout << "asdfgh" <<'|'<< setw(4) << "asdfgh" << '|'
            << setw(8) << "asdfgh" << '|' << "asdfgh" << "|\n";
// results:
    123456|123456|  123456|123456|
    1234.56|1234.56| 1234.56|1234.56|
    asdfgh|asdfgh|  asdfgh|asdfgh|
```

Observation

- This kind of detail is what you need textbooks, manuals, references, online support, etc. for
  - You **always** forget some of the details when you need them

A file



- At the fundamental level, a file is a sequence of bytes numbered from 0 upwards
- Other notions can be supplied by programs that interpret a “file format”
  - For example, the 6 bytes "123.45" might be interpreted as the floating-point number 123.45

File open modes

- By default, an **ifstream** opens its file for reading
- By default, an **ofstream** opens its file for writing.
- Alternatives:
  - **ios_base::app** *// append (i.e., add to the end of the file)*
  - **ios_base::ate** *// "at end" (open and seek to end)*
  - **ios_base::binary** *// binary mode – beware of system specific behavior*
  - **ios_base::in** *// for reading*
  - **ios_base::out** *// for writing*
  - **ios_base::trunc** *// truncate file to 0-length*
- A file mode is optionally specified after the name of the file:
  - **ofstream of1(name1);** *// defaults to ios_base::out*
  - **ifstream if1(name2);** *// defaults to ios_base::in*
  - **ofstream ofs(name, ios_base::app);** *// append rather than overwrite*
  - **fstream fs("myfile", ios_base::in|ios_base::out);** *// both in and out*

Text vs. binary files

123 as characters:

12345 as characters:

123 as binary:

12345 as binary:

123456 as characters:

123 456 as characters:

In binary files, we use sizes to delimit values

In text files, we use separation/termination characters

Text vs. binary

- Use text when you can
  - You can read it (without a fancy program)
  - You can debug your programs more easily
  - Text is portable across different systems
  - Most information can be represented reasonably as text
- Use binary when you must
  - E.g. image files, sound files

Binary files

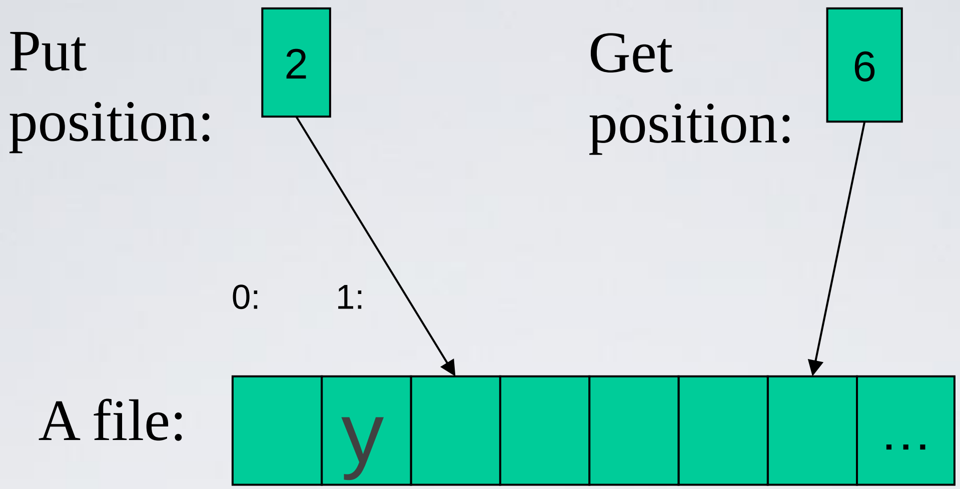
```
int main()
    // use binary input and output
{
    cout << "Please enter input file name\n";
    string name;
    cin >> name;
    ifstream ifs(name.c_str(),ios_base::binary);        // note: binary
    if (!ifs) error("can't open input file ", name);

    cout << "Please enter output file name\n";
    cin >> name;
    ofstream ofs(name.c_str(),ios_base::binary);        // note: binary
    if (!ofs) error("can't open output file ",name);

    // "binary" tells the stream not to try anything clever with the bytes
```

Binary files

```
vector<int> v;

// read from binary file:
int i;
while (ifs.read(as_bytes(i),sizeof(int)))          // note: reading bytes
        v.push_back(i);

// ... do something with v ...

// write to binary file:
for(int i=0; i<v.size(); ++i)
        ofs.write(as_bytes(v[i]),sizeof(int));     // note: writing bytes
return 0;
}
```

*// for now, treat **as_bytes()** as a primitive*

Positioning in a filestream




```
fstream fs(name.c_str());            // open for input and output

// …
fs.seekg(5);      // move reading position ('g' for 'get') to 5 (the 6th character)
char ch;
fs>>ch;           // read and increment reading position
cout << "character[6] is " << ch << '(' << int(ch) << ")\n";
fs.seekp(1);      // move writing position ('p' for 'put') to 1 (the 2nd character)
fs<<'y';          // write and increment writing position
```

Positioning

- Whenever you can
  - Use simple streaming
    - Streams/streaming is a very powerful metaphor
    - Write most of your code in terms of "plain" **istream** and **ostream**
  - Positioning is far more error-prone
    - Handling of the end of file position is system dependent and basically unchecked

String streams

A **stringstream** reads/writes from/to a **string** rather than a file or a keyboard/screen

```
double str_to_double(string s)
        // if possible, convert characters in s to floating-point value
{
   istringstream is(s);      // make a stream so that we can read from s
   double d;
   is >> d;
   if (!is) error("double format error");
   return d;
}

double d1 = str_to_double("12.4");                     // testing
double d2 = str_to_double("1.34e-3");
double d3 = str_to_double("twelve point three");   // will call error()
```

String streams

- Very useful for
  - formatting into a fixed-sized space (think GUI)
  - for extracting typed objects out of a string

Type vs. line

- Read a string

```
string name;
cin >> name;                    // input: Dennis Ritchie
cout << name << '\n';           // output: Dennis
```

- Read a line

```
string name;
getline(cin,name);   // input: Dennis Ritchie
cout << name << '\n';           // output: Dennis Ritchie
// now what?
// maybe:
istringstream ss(name);
ss>>first_name;
ss>>second_name;
```

Characters

- You can also read individual characters

```
char ch;
while (cin>>ch) {        // read into ch, skipping whitespace characters
   if (isalpha(ch)) {
                  // do something
   }
}

while (cin.get(ch)) {    // read into ch, don't skip whitespace characters
   if (isspace(ch)) {
                  // do something
   }
   else if (isalpha(ch)) {
                  // do something else
   }
}
```

Character classification functions

- If you use character input, you often need one or more of these (from header **<cctype>**):

  - **isspace(c)** *// is **c** whitespace? (' ', '\t', '\n', etc.)*
  - **isalpha(c)** *// is **c** a letter? ('a'..'z', 'A'..'Z') note: not '_'*
  - **isdigit(c)** *// is **c** a decimal digit? ('0'.. '9')*
  - **isupper(c)** *// is **c** an upper case letter?*
  - **islower(c)** *// is **c** a lower case letter?*
  - **isalnum(c)** *// is **c** a letter or a decimal digit?*

Line-oriented input

- Prefer >> to **getline()**
  - i.e. avoid line-oriented input when you can

- People often use **getline()** because they see no alternative
  - But it often gets messy

- When trying to use **getline()**, you often end up
  - using >> to parse the line from a **stringstream**
  - using **get()** to read individual characters

Next lecture

- Graphical output
  - Creating a GUI window with various figures
  - Drawing graphs

Acknowledgements

Bjarne Stroustrup

Programming -- Principles and Practice Using C++

http://www.stroustrup.com/Programming/

Thank you!

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης