Customizing I/O

Michael Nowak

Texas A&M University

Acknowledgement: Lecture slides based on those created by Bjarne Stroustrup for use with his textbook

Notes

Overview

Output formats
Formatting output
- Integer output
- Floating-point output
  - defaultfloat format
  - scientific format
  - fixed format

Output fields
- Field width
- Field fill
- Field adjustment

Line-oriented input
- Type vs. line
- Mixing formatted input with line-oriented input - be careful
- Prefer formatted input

Character-oriented input
- Character classification functions

References

Notes

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 110 mean?
    - What does 123,456 mean?
    - What does (123) mean?
- The world (of output formats) is far more particular than you could possibly imagine

Notes

Overview

Output formats

Formatting output

- Integer output
- Floating-point output
  - defaultfloat format
  - scientific format
  - fixed format

Output fields

- Field width
- Field fill
- Field adjustment

Line-oriented input

- Type vs. line
- Mixing formatted input with line-oriented input - be careful
- Prefer formatted input

Character-oriented input

- Character classification functions

References

Notes

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)

Notes

Overview

Output formats

Formatting output

- Integer output
- Floating-point output
  - defaultfloat format
  - scientific format
  - fixed format

Output fields

- Field width
- Field fill
- Field adjustment

Line-oriented input

- Type vs. line
- Mixing formatted input with line-oriented input - be careful
- Prefer formatted input

Character-oriented input

- Character classification functions

References

Notes

Formatting output

- Output formatting is controlled by a set of flags and integer values for a given stream
  - Integral output
  - Floating-point output
  - Output fields
  - Field adjustment
- Manipulators
  - Manipulators allow us to manipulate the state of a stream; they are inserted between the objects being read or written
  - Most manipulators are `sticky` : they are set and permanent until changed

```
#include<iomanip>
```

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Integer output

- Decimal (base-10)
  - Manipulator: `std::dec` (sticky)
    `cout << dec << 1234;` ⟶ `1234`
- Octal (base-8)
  - Manipulator: `std::oct` (sticky)
    `cout << oct << 1234;` ⟶ `2322`
- Hexadecimal (base-16)
  - Manipulator: `std::hex` (sticky)
    `cout << hex << 1234;` ⟶ `4d2`
- Showbase
  - Manipulator: `std::showbase` (sticky)
    `cout << showbase << oct << 1234;` ⟶ `02322`
    `cout << showbase << hex << 1234;` ⟶ `0x4d2`
- Noshowbase
  Manipulator: `std::noshowbase` (sticky)

Notes

Integer output ex1

```
// simple test:
cout << dec << 1234 << "\t(decimal)\n"
     << hex << 1234 << "\t(hexadecimal)\n"
     << oct << 1234 << "\t(octal)\n";

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

Notes

Integer output ex2

```
// simple test:
cout << 1234 << '\t'
     << hex  << 1234 << '\t'
     << oct  << 1234 << '\n';
cout << 1234 << '\n';

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

Notes

Integer output ex3

```
// simple test:
cout << 1234  << '\t'
     << hex  << 1234 << '\t'
     << oct  << 1234 << endl;
cout << showbase << dec;
cout << 1234 << '\t'
     << hex  << 1234 << '\t'
     << oct  << 1234 << '\n';

// results:
1234    4d2 2322
1234    0x4d2   02322
```

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Floating-point output

- Floating-point output formatting is controlled by its `format` and `precision`
- The floating-point value being output is rounded to give the best approximation that can be printed given the specified `precision` in the chosen `format`
- The default `precision` is six digits

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

`defaultfloat` format

- The `defaultfloat` format lets the implementation choose the format (`fixed` or `scientific`) that presents a value in the style that best preserves the value in the specified `precision`
- For `defaultfloat`, precision specifies the maximum number of digits
  - We can specify the `precision` using the `setprecision()` manipulator (sticky)
- `cout << defaultfloat << 1234.567;` $\longrightarrow$ `1234.57`
- `cout << defaultfloat << 1234567.0;` $\longrightarrow$ `1.23457e+006`

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

`scientific` format

- The `scientific` format presents a value with one digit before a decimal point followed by an exponent
- The `precision` is the number *n* of digits after the decimal point
- `cout << scientific << 1234.56789;` $\longrightarrow$ `1.234568e+03`
- `cout << scientific << setprecision(3) << 1234.56789;` $\longrightarrow$ `1.235e+03`

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

`fixed` format

- The `fixed` format presents a floating-point value as an integer followed by a decimal point and a fractional part
- The `precision` is the number of digits after the decimal point
- `cout << fixed << 1234.56789` ⟶ `1234.567890`
- `cout << fixed << setprecision(3) << 1234.56789` ⟶ `1234.568`

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Output fields

- Frequently, we would like to fill a specific number of character spaces on an output line with text
  - E.g., we want exactly *n*-characters and not fewer (and more only if the text does not fit)
- We can specify a field width, and a character to be used if padding is needed, for a value being output

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Field width

- We can specify the minimum number of characters to be used in an output field
- Manipulator: `std::setw()` (not-sticky)
  - By default, the text is right-aligned in the output field

```
cout << setw(4) << 1; ⟶ ␣␣␣1
cout << setw(8) << "Michael"; ⟶ ␣Michael
```

Notes

Overview

Output formats
Formatting output
  Integer output
  Floating-point output
    defaultfloat format
    scientific format
    fixed format
Output fields
  Field width
  Field fill
  Field adjustment
Line-oriented input
  Type vs. line
  Mixing formatted input with line-oriented input - be careful
  Prefer formatted input
Character-oriented input
  Character classification functions
References

Notes

Field fill

- We can specify the “padding” or “filler” character of an output field
- Manipulator: `std::setfill()` (sticky)

  `cout << setw(4) << setfill('*') << 1;` $\longrightarrow$ `***1`

Notes

Overview

Output formats
Formatting output
  Integer output
  Floating-point output
    defaultfloat format
    scientific format
    fixed format
Output fields
  Field width
  Field fill
  Field adjustment
Line-oriented input
  Type vs. line
  Mixing formatted input with line-oriented input - be careful
  Prefer formatted input
Character-oriented input
  Character classification functions
References

Notes

Field adjustment

- We can adjust characters within a filed
  - `right` (non-sticky) adjustment (default), which right-aligns the characters within a field
    `cout << setw(4) << 1;` ⟶ `␣␣␣1`
  - `left` (non-sticky) adjustment, which left-aligns the characters within a field
    `cout << setw(4) << << setfill('*') << right << 1;` ⟶ `1***`
  - `internal` (non-sticky) adjustment, which places fill characters between the sign and the value
    `cout << setw(4) << << setfill('.') << internal << -1;` ⟶ `-..1`

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Type vs. line

- Read a string:

```
string name;
cin >> name;
// input: Bjarne Stroustrup
cout << name << endl;
// output: Bjarne
```

- Read a line:

```
string name;
getline(cin,name);
// input: Bjarne Stroustrup
cout << name << endl;
// output: Bjarne Stroustrup
/* now what? */
/* maybe... */
istringstream ss(name);
ss >> first_name;
ss >> second_name;
```

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Mixing formatted input with line-oriented input - be careful

```
int yob; string name;
cin >> yob;
// input: 1950
getline(cin,name);
cout << yob << '\t' << name << endl;
Output: 1950
```

- (cin) reads formatted input and delimits on white-spaces
- This means that there is still a linefeed left-over in the input buffer from the character return when I entered `1950` `[Return]`
- It is that character return that is read by `getline(cin,name)`
- Meaning that `getline` does not `block` for data from standard input

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Prefer formatted input

- Prefer formatted input `>>` to line-oriented input `getline()`
  - i.e. avoid line-oriented input when you can
- People often use `getline()` because they see no alternative
  - But it easily 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

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Character-oriented input

- You can also read individual characters:

```
for (char ch; cin >> ch; )
{
    if (isalpha(ch)) {
        // do something
    }
}

for (char ch; cin.get(ch); )
{
    characters
    if (isspace(ch)) {
        // do something
    } else if (isalpha(ch)) {
        // do something else
    }
}
```

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

Character classification functions

- If you use character-oriented 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 ('A'..'Z')?
  - `islower(c)`
    - is `c` a lower case letter ('a'..'z')?
  - `isalnum(c)`
    - is `c` a letter or a decimal digit ('a'..'z', 'A'..'Z', '0'..'9')?

Notes

Overview

Output formats
Formatting output
Integer output
Floating-point output
defaultfloat format
scientific format
fixed format
Output fields
Field width
Field fill
Field adjustment
Line-oriented input
Type vs. line
Mixing formatted input with line-oriented input - be careful
Prefer formatted input
Character-oriented input
Character classification functions
References

Notes

References

- Lippman, B., Lajoie, Josee, & Moo, B. E. (2016). *C++ primer* (5th ed.). Addison-Wesley.
- Stroustrup, B. (2014). *Programming: principles and practice using C++* (2nd ed.). Addison-Wesley.

Notes

Notes