Exercise 10 –Input Streams, BNF, Graded Homework

Informatik I für Mathematiker und Physiker (HS 2015)

Yeara Kozlov

Slides courtesy of Virag Varga

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Agenda

- HW #8 recap
- Input streams
- Backus-Naur-Form
- Hints for homework
- Graded homework
  - Hexadecimal representation

Agenda

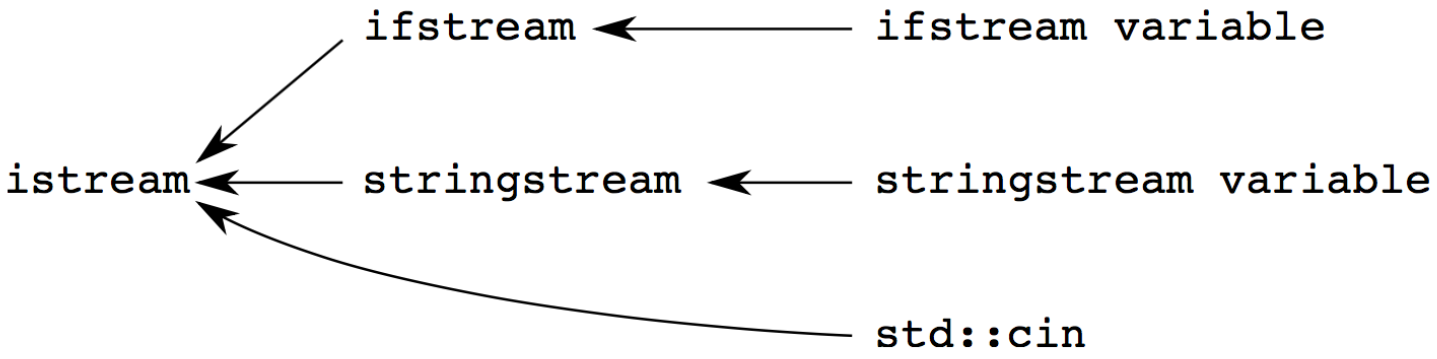
- **HW #8 recap**
- Input streams
- Backus-Naur-Form
- Hints for homework
- Graded homework
  - Hexadecimal representation

Agenda

- HW #8 recap
- **Input streams**
- Backus-Naur-Form
- Hints for homework
- Graded homework
  - Hexadecimal representation

Input Streams

- Input stream objects can read and interpret input from



using stringstream

```
#include <sstream>

std::stringstream nbrs("45 10 33");

// like std::cin --> can read as int

int i;
nbrs >> i;    // read 45 (as int)
              // nbrs is now: 10 33

// like std::cin --> can also read as other types
char c;
nbrs >> c;    // read 1 (as char)
nbrs >> c;    // read 0 (as char)
              // nbrs is now: 33

double d;
nbrs >> d;    // read 33 (as double)
```

using stringstream

```
#include <iostream>
#include <sstream>
#include <fstream>      // for std::ifstream
#include <iostream>

void output_istream (std::istream& is) {
  char c;
  is >> std::noskipws; // do not ignore whitespaces
  while (is >> c)  std::cout << c;
}

int main () {
 std::stringstream str ("Hello");
 output_istream(str); // str is now empty!

 output_istream(std::cin); // outputs everything the user enters right away

 std::ifstream file_stream ("my_file.txt");
 output_istream(file_stream); // outputs the whole file to the terminal

 return 0;
}
```

Stream Exercise

Write a function `rev_out` (see template below) which outputs the contents of an `istream` in reverse order using recursion.

```
#include <iostream>
#include <sstream>

// POST: output the content of is in reverse order to
//       std::cout, and removed it from is.
void rev_out (std::istream& is)
{
  // your code
}

int main () {
  std::stringstream input ("abcdefghijklmno");
  rev_out (input);
  return 0;
}
```

Stream Exercise

Other solutions are of course also possible.

```
#include <iostream>
#include <sstream>

// POST: output the content of is in reverse order to
//       std::cout, and removed it from is.
void rev_out (std::istream& is)
{
  char val;
  if (is >> val) {
    rev_out(is);
    std::cout << val;
  }
}

int main () {
  std::stringstream input ("abcdefghijklmno");
  rev_out (input);
  return 0;
}
```

Agenda

- HW #8 recap
- Input streams
- **Backus-Naur-Form**
- Hints for homework
- Graded homework
  - Hexadecimal representation

BNF

- A recursive way to describe a set of sequences.
- Sequences are obtained by concatenating elements of a predefined alphabet according to predefined rules.

- To recap: a set of rules which can be used to generate every allowed sequence.

BNF - Example

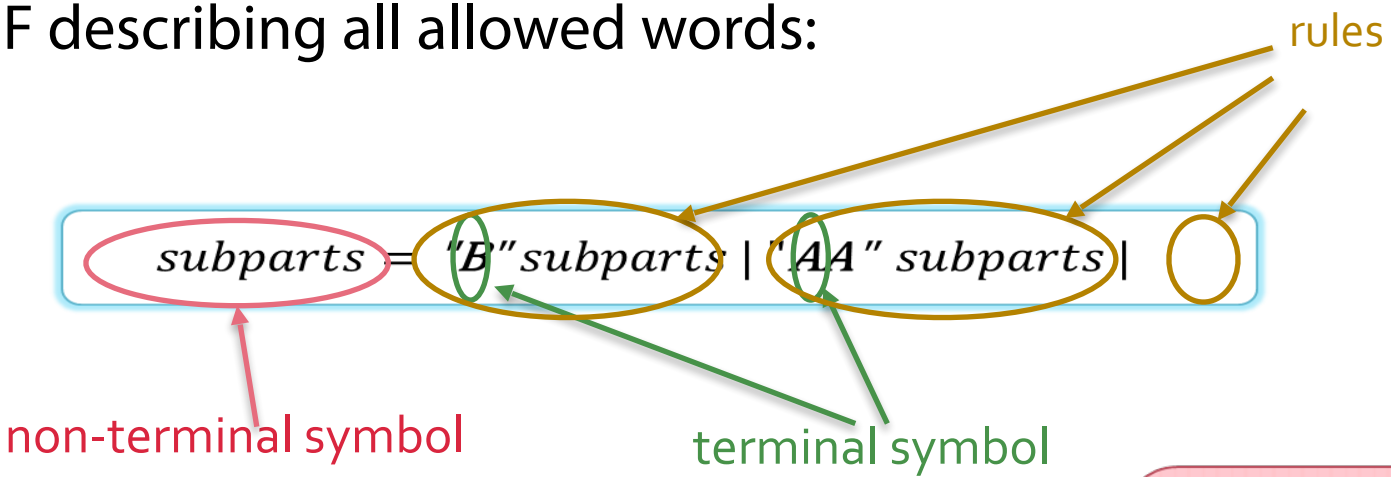
- Alphabet: $\{A, B\}$
- Rules: A *can only appear in pairs like* AA
  - Example of valid word: $BBBAABAA$

- BNF describing all allowed words:

$$subparts = "B" subparts \mid "AA" subparts \mid$$

BNF

- BNF describing all allowed words:



  - 1 rules with 3 alternatives
  - 2 terminal symbols ("*A*" and "*B*")
    - the substitution-recursion stops
  - 1 non-terminal symbol (*subparts*)
    - has to be substituted further

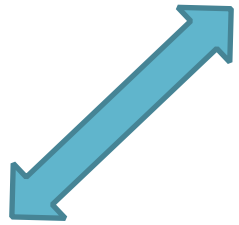
How to obtain: ***BBBAABAA*** ?

BNF - Exercise from Exam, Winter 15

- Given this BNF:

```
list =  "{" items "}"
items = item | item "," items
item = text | text list
```

- text is an arbitrary (possibly empty) sequence of characters not containing { } ;
  - A valid sentence for example:



- letter
  - upper case
    - A
    - B
    - C
  - lower case
    - a
    - b
    - c

```
{letter {upper case {A,B,C}, lower case {a,b,c}}}
```

BNF - Which lists are valid?

```
list =  "{" items "}"
items = item | item "," items
item = text | text list
```

- **text** is an arbitrary (possibly empty) sequence of characters not containing { } ;

a) {Languages {French,English,Swiss German}} — **true**

b) {Food {Health Junk}} — **true**

c) {Darth {Vader} {Maul}} — **false**

d) Table of Contents {Section 1, Section 2 {Subsection 2.1}} — **false**

e) *What is the shortest valid list?* — **{}**

Nesting Depth

- The nesting depth of an arithmetic expression counts how many pairs of parentheses enclose the innermost number in the expression.

- $5 \longrightarrow 0$
- $(3 + 4) * (5 / 6) \longrightarrow 1$
- $((3 + 4) * (5 / 6)) \longrightarrow 2$

Nesting Depth in BNF

$$((3 + 4) * (5/6)) + (-7)$$

number, (expression),
-number, -(expression)

factor * factor, factor,
factor * factor / factor, …

term + term, term, term
– term, …

```
factor = "(" expression ")"
       | "–" factor
       | unsigned_number.
```

```
term = factor
     | factor "*" term
     | factor "/" term.
```

```
expression = term
           | term "+" expression
           | term "–" expression.
```

Nesting Depth in BNF

- d(x) $\Longrightarrow$ nesting depth of 'x'

$$((3 + 4) * (5/6)) + (-7)$$

```
factor = "(" expression ")"
       | "-" factor
       | unsigned_number.
```

- d(f) = d(e) + 1
- d(f) = d(f')
- 0

```
term = factor
     | factor "*" term
     | factor "/" term.
```

- d(t) = d(f)
- d(t) = max (d(f), d(t'))
- d(t) = max (d(f), d(t'))

```
expression = term
           | term "+" expression
           | term "-" expression.
```

- d(e) = d(t)
- max..
- max..

Graded Homework

- Big homework sheet - six pages

- Create a CPU simulator
  - Disassembly
    - Hexadecimal numbers need to be decoded into machine instructions
  - Simulation
    - Implement a processor which processes said machine instructions

Graded Homework

- Two weeks - start early
- Read the instructions **carefully**
  - Describes everything you need to implement
- We can only give very limited help.

Hexadecimal Representation

Hexadecimal Representation

- The hexadecimal digits…

hexadec	decimal	bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

hexadec	decimal	bin
8	8	1000
9	9	1001
a	10	1010
b	11	1011
c	12	1100
d	13	1101
e	14	1110
f	15	1111

Hexadecimal Representation

- … and how to count them.

hexadecimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
decimal:	0	1	2	3	4	5	6	7	8	9

Hexadecimal Representation

- … and how to count them.

hexadecimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**
decimal:	0	1	2	3	4	5	6	7	8	9	10

Hexadecimal Representation

- … and how to count them.

hexadecimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**
decimal:	0	1	2	3	4	5	6	7	8	9	10	11

Hexadecimal Representation

- … and how to count them.

hexadecimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**
decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12

Hexadecimal Representation

- … and how to count them.

hexadecimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**
decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Hexadecimal Representation

- … and how to count them.

hexadecimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**
decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Hexadecimal Representation

- … and how to count them.

hexadecimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Hexadecimal Representation

- … and how to count them.

hexadecimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**	**10**
decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Hexadecimal Representation

- … and how to count them.

hexadecimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**	**10**	**11**	**12**	**13**
decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

hexadecimal:	**14**	**15**	**16**	**17**	**18**	**19**	**1a**	**1b**	**1c**	**1d**	**1e**	**1f**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**…**
decimal:	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	…

Hexadecimal Representation

- Why hexadecimal?
  - $16 = 2 * 2 * 2 * 2$
  - Hexadecimal numbers can be used to express binary numbers in a shorter, easier to read form.
- One hexadecimal digit represents a 4 bit (4 bit is half a byte, sometimes called a nibble) binary number.

Hexadecimal Representation

- A 32 bit binary number (for example, unsigned int) can be expressed as a number between `0x00000000` and `0xffffffff`.

```
unsigned int a = 23;
unsigned int b = 0x00000017;
if (a == b)  std::cout << "a equal to b\n"; // this is output

unsigned int c = 0x0c38d220;
unsigned int d = 28121760288;
if (c == d)  std::cout << "c equal to d\n"; // this is also output
```

Reading Hex Input

- By default, numbers are in decimal representation, and `a,b,c...` are interpreted as characters.
- Use `std::hex` to change this interpretation:

```
// assume the input is:  20 20 20
unsigned int i;
std::cin >> i;
std::cout << i << " "; // output: 20
std::cin >> std::hex;
std::cin >> i;
std::cout << i << " "; // output: 32 (same as 0x00000020)
std::cin >> std::dec; // back to decimal mode
std::cin >> i;
std::cout << i << "\n"; // output: 20
```

Exercise - Hex Conversion

- Convert the following numbers from decimal to hex:
  - 2
  - 12
  - 49
  - 108

Exercise - Hex Conversion

- Convert the following numbers from decimal to hex:
  - 2
  - 12
  - 49
  - 108