CS 460

Programming Languages

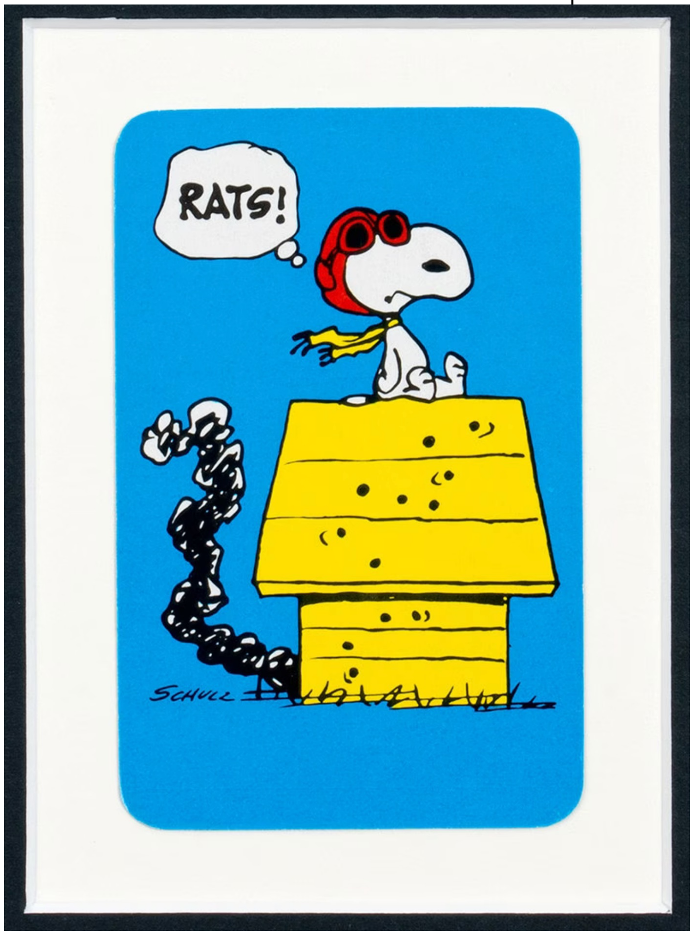
Fall 2021

Dr. Watts

(9 October 2023)

Exercise 2

- Will be revisiting this as Exercise 5
- Issues
  - Testing
  - C++ input protocols
- Preliminary Exercise
  - Will be posted soon – not due for quite a while.



Project 1

- Questions? (By email or on paper)

Project 1

- Regarding the white leading whitespaces in the .lst files before the line number. Does our output .lst file have to match the format exactly? Did you use \t to tab it or what width is that set to?
- .lst files do not need to match.
- .p1 files need to match. (White space not important.)

Project 1

- How should the .lst and and.p1 files look for an input such as:

  .-123.43 12/.5 ./

```
Input file: ha.pl460
   1: .-123.43 12/.5 ./
Error at 1,1: Unexpected '.' found
Error at 1,12: Unexpected '12/' found
Error at 1,16: Unexpected '.' found
3 errors found in input file
```

```
ERROR_T         .
NUMLIT_T        -123.43
ERROR_T         12/
NUMLIT_T        .5
ERROR_T         .
DIV_T           /
EOF_T
```

IEEE Single Precision (float 32)

- The IEEE single precision floating point standard representation requires a 32 bit word, which may be represented as numbered from 0 to 31, left to right. The first bit is the sign bit, S, the next eight bits are the exponent bits, 'E', and the final 23 bits are the fraction 'F':

```
S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
0 1      8 9                    31
```

- The value V represented by the word may be determined as follows:
  - If E=255 and F is nonzero, then V=NaN ("Not a number")
  - If E=255 and F is zero and S is 1, then V=-Infinity
  - If E=255 and F is zero and S is 0, then V=Infinity
  - If 0<E<255 then V=(-1)**S * 2 ** (E-127) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
  - If E=0 and F is nonzero, then V=(-1)**S * 2 ** (-126) * (0.F) These are "unnormalized" values.
  - If E=0 and F is zero and S is 1, then V=-0
  - If E=0 and F is zero and S is 0, then V=0
- In particular,
  - `0 00000000 00000000000000000000000 = 0`
  - `1 00000000 00000000000000000000000 = -0`
  - `0 11111111 00000000000000000000000 = Infinity`
  - `1 11111111 00000000000000000000000 = -Infinity`
  - `0 11111111 00000100000000000000000 = NaN`
  - `1 11111111 00100010001001010101010 = NaN`
  - `0 10000000 00000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2`
  - `0 10000001 10100000000000000000000 = +1 * 2**(129-127) * 1.101 = 6.5`
  - `1 10000001 10100000000000000000000 = -1 * 2**(129-127) * 1.101 = -6.5`
  - `0 00000001 00000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)`
  - `0 00000000 10000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)`
  - `0 00000000 00000000000000000000001 = +1 * 2**(-126) * 0.00000000000000000000001 = 2**(-149) (Smallest positive value)`

IEEE Double Precision (float64)

- The IEEE double precision floating point standard representation requires a 64 bit word, which may be represented as numbered from 0 to 63, left to right. The first bit is the sign bit, S, the next eleven bits are the exponent bits, 'E', and the final 52 bits are the fraction 'F':

```
S EEEEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
0 1        11 12                                                63
```

- The value V represented by the word may be determined as follows:
- If E=2047 and F is nonzero, then V=NaN ("Not a number")
- If E=2047 and F is zero and S is 1, then V=-Infinity
- If E=2047 and F is zero and S is 0, then V=Infinity
- If 0<E<2047 then V=(-1)**S * 2 ** (E-1023) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E=0 and F is nonzero, then V=(-1)**S * 2 ** (-1022) * (0.F) These are "unnormalized" values.
- If E=0 and F is zero and S is 1, then V=-0
- If E=0 and F is zero and S is 0, then V=0

IEEE Quadruple Precision (float128)

- 1 bit sign
- 15 bit exponent
- 112 bit fraction

Half Precision (float16)

- Half precision floating point representation requires a 16 bit word, which may be represented as numbered from 0 to 15, left to right. The first bit is the sign bit, S, the next five bits are the exponent bits, 'E', and the final 10 bits are the fraction 'F':

```
S EEEEE FFFFFFFFFF
0 1   5 6        15
```

- The value V represented by the word may be determined as follows
- Sign
  - Sign = 0 is positive
  - Sign = 1 is negative
- Exponent
  - Biased (15)
  - 00000 – 11111
- Fraction

Half Precision Example 1

- Example: 125.25
  - Whole number 125_{10} == $?_2$
  - Fraction $.25_{10}$ == $?_2$

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value																

Half Precision Example 2

- Example: 123000.0
  - Whole number $123000_{10} == ?_2$
  - Fraction $.0_{10} == ?_2$

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value																

Half Precision Example 3

- Example: 0.21875
  - Whole number 0_{10} == $?_2$
  - Fraction $.21875_{10}$ == $?_2$

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value																

Half Precision Example 4

- Example: 5.20
  - Whole number 5_{10} == $?_2$
  - Fraction $.20_{10}$ == $?_2$

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value																

Half Precision Example 5

- Example: 12.20
  - Whole number $12_{10} == ?_2$
  - Fraction $.20_{10} == ?_2$

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value																

Half Precision Example 6

- Example: 25.20
  - Whole number $25_{10} == ?_2$
  - Fraction $.20_{10} == ?_2$

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value																

Half Precision Example 7

- Example: 37.20
  - Whole number $37_{10} == ?_2$
  - Fraction $.20_{10} == ?_2$

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value																

Let's go the other way!

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0

Let’s go the other way!

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value	1	1	0	1	1	1	0	0	1	1	0	0	1	1	0	1

Some Half-precision values

Binary	Hex	Value	Notes
0 00000 0000000000	0000	0	
0 00000 0000000001	0001	$2^{-14} \times (0 + \frac{1}{1024}) \approx 0.000000059604645$	smallest positive subnormal number
0 00000 1111111111	03ff	$2^{-14} \times (0 + \frac{1023}{1024}) \approx 0.000060975552$	largest subnormal number
0 00001 0000000000	0400	$2^{-14} \times (1 + \frac{0}{1024}) \approx 0.00006103515625$	smallest positive normal number
0 01101 0101010101	3555	$2^{-2} \times (1 + \frac{341}{1024}) \approx 0.33325195$	nearest value to 1/3
0 01110 1111111111	3bff	$2^{-1} \times (1 + \frac{1023}{1024}) \approx 0.99951172$	largest number less than one
0 01111 0000000000	3c00	$2^{0} \times (1 + \frac{0}{1024}) = 1$	one
0 01111 0000000001	3c01	$2^{0} \times (1 + \frac{1}{1024}) \approx 1.00097656$	smallest number larger than one
0 11110 1111111111	7bff	$2^{15} \times (1 + \frac{1023}{1024}) = 65504$	largest normal number
0 11111 0000000000	7c00	∞	infinity
1 00000 0000000000	8000	-0	
1 10000 0000000000	c000	-2	
1 11111 0000000000	fc00	$-\infty$	negative infinity

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main ()
{
        float value = 0;
        do
        {
                cout << "Please enter a floating point value (0 to quit): ";
                cin >> value;
                cout << "  value: " << fixed << showpoint << setprecision (30)
                     << value << endl;
                float rounded = (round (value * 100)) / 100.0;
                cout << "rounded: " << fixed << showpoint << setprecision (30)
                     << rounded << endl;
        } while (value != 0);
        return 0;
}
```

Other Numeric Types

- Arbitrary precision
- Fraction type
- User defined types

String types

- C style strings
- C++ strings

Composite Data Types

- Array
- Struct
- Class
- Union

Contents

Chapter 1 Preliminaries 25

1.1 Reasons for Studying Concepts of Programming Languages 26
1.2 Programming Domains 29
1.3 Language Evaluation Criteria 30
1.4 Influences on Language Design 41
1.5 Language Categories 44
1.6 Language Design Trade-Offs 45
1.7 Implementation Methods 46
1.8 Programming Environments 53
Summary • Review Questions • Problem Set 54

Chapter 2 Evolution of the Major Programming Languages 57

2.1 Zuse's Plankalkül 60
2.2 Pseudocodes 61
2.3 The IBM 704 and Fortran 64
2.4 Functional Programming: Lisp 69
2.5 The First Step Toward Sophistication: ALGOL 60 74
2.6 Computerizing Business Records: COBOL 80
2.7 The Beginnings of Timesharing: Basic 85
Interview: **ALAN COOPER—User Design and Language Design** 88
2.8 Everything for Everybody: PL/I 90
2.9 Two Early Dynamic Languages: APL and SNOBOL 93
2.10 The Beginnings of Data Abstraction: SIMULA 67 94
2.11 Orthogonal Design: ALGOL 68 95
2.12 Some Early Descendants of the ALGOLs 97

2.13 Programming Based on Logic: Prolog 101
2.14 History's Largest Design Effort: Ada 103
2.15 Object-Oriented Programming: Smalltalk 107
2.16 Combining Imperative and Object-Oriented Features: C++ 109
2.17 An Imperative-Based Object-Oriented Language: Java 112
2.18 Scripting Languages 115
2.19 The Flagship .NET Language: C# 122
2.20 Markup-Programming Hybrid Languages 124
Summary • Bibliographic Notes • Review Questions • Problem Set • Programming Exercises 126

Chapter 3 Describing Syntax and Semantics 133

3.1 Introduction 134
3.2 The General Problem of Describing Syntax 135
3.3 Formal Methods of Describing Syntax 137
3.4 Attribute Grammars 152
History Note 152
3.5 Describing the Meanings of Programs: Dynamic Semantics 158
History Note 166
Summary • Bibliographic Notes • Review Questions • Problem Set 179

Chapter 4 Lexical and Syntax Analysis 185

4.1 Introduction 186
4.2 Lexical Analysis 187
4.3 The Parsing Problem 195
4.4 Recursive-Descent Parsing 199
4.5 Bottom-Up Parsing 207
Summary • Review Questions • Problem Set • Programming Exercises 215

Chapter 5 Names, Bindings, and Scopes 221

5.1 Introduction 222
5.2 Names 223

History Note 223

5.3 Variables 224

5.4 The Concept of Binding 227

5.5 Scope 235

5.6 Scope and Lifetime 246

5.7 Referencing Environments 247

5.8 Named Constants 248

Summary • Review Questions • Problem Set • Programming Exercises 251

Chapter 6 Data Types 259

6.1 Introduction 260

6.2 Primitive Data Types 262

6.3 Character String Types 266

History Note 267

6.4 Enumeration Types 271

6.5 Array Types 274

History Note 275

History Note 275

6.6 Associative Arrays 285

Interview: **ROBERTO IERUSALIMSCHY—Lua** 286

6.7 Record Types 289

6.8 Tuple Types 292

6.9 List Types 294

6.10 Union Types 296

6.11 Pointer and Reference Types 299

History Note 302

6.12 Type Checking 311

6.13 Strong Typing 312

6.14 Type Equivalence 313

6.15 Theory and Data Types 317

Summary • Bibliographic Notes • Review Questions • Problem Set • Programming Exercises 319