

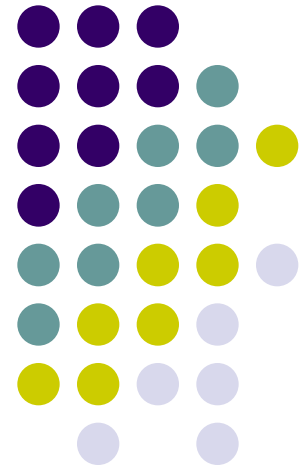
# CS 460

Programming Languages

Fall 2021

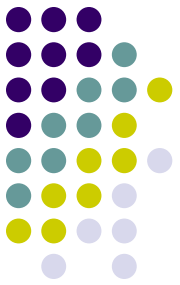
Dr. Watts

(11 October 2023)

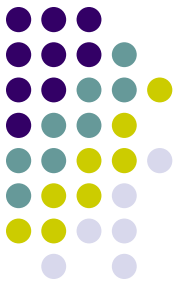


# Project 1

- Questions? (By email or on paper)



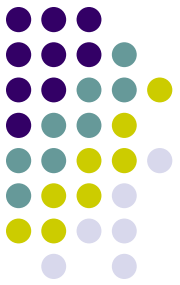
# Half Precision (float16)



- 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

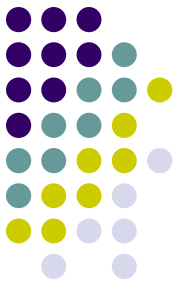


# Half Precision Example 4

- Example: 5.20
- Whole: 5 : 101
- Fraction: .2 : 001100110011001100
- Unnormalized: 101.001100110011001100
- Exponent: 2
- Normalized: 1.01001100110011001100
- Next is 0

5.2

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

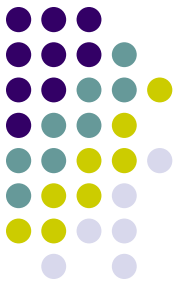


# Half Precision Example 5

- Example: 12.20
- Whole: 12 : 1100
- Fraction: .2 : 001100110011001100
- Unnormalized: 1100.001100110011001100
- Exponent: 3
- Normalized: 1.100001100110011001100
- Next is 1

12.2

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

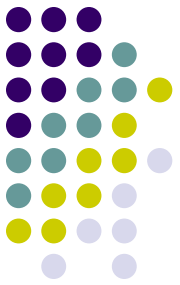


# Half Precision Example 6

- Example: 25.20
- Whole: 25 : 11001
- Fraction: .2 : 001100110011001100
- Unnormalized: 11001.001100110011001100
- Exponent: 4
- Normalized: 1.1001001100110011001100
- Next is 1

25.2

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



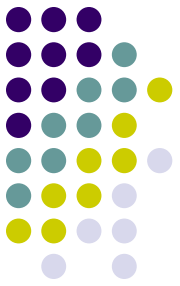
# Half Precision Example 7

- Example: 37.20
- Whole: 37 : 100101
- Fraction: .2 : 001100110011001100
- Unnormalized: 100101.001100110011001100
- Exponent: 5
- Normalized: 1.00101001100110011001100
- Next is 0

37.2

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

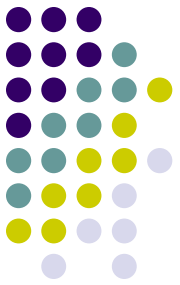
# Let's go the other way!



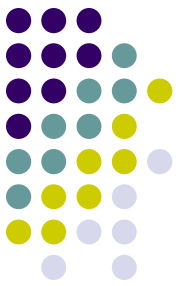
Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0



# 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	0	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 00000000000	0000	0	
0 00000 00000000001	0001	$2^{-14} \times (0 + \frac{1}{1024}) \approx 0.000000059604645$	smallest positive subnormal number
0 00000 11111111111	03ff	$2^{-14} \times (0 + \frac{1023}{1024}) \approx 0.000060975552$	largest subnormal number
0 00001 00000000000	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 11111111111	3bff	$2^{-1} \times (1 + \frac{1023}{1024}) \approx 0.99951172$	largest number less than one
0 01111 00000000000	3c00	$2^0 \times (1 + \frac{0}{1024}) = 1$	one
0 01111 00000000001	3c01	$2^0 \times (1 + \frac{1}{1024}) \approx 1.00097656$	smallest number larger than one
0 11110 11111111111	7bff	$2^{15} \times (1 + \frac{1023}{1024}) = 65504$	largest normal number
0 11111 00000000000	7c00	$\infty$	infinity
1 00000 00000000000	8000	-0	
1 10000 00000000000	c000	-2	
1 11111 00000000000	fc00	$-\infty$	negative infinity

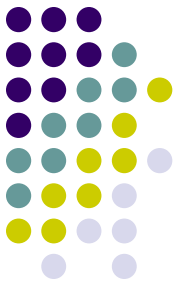
```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

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;
}
```



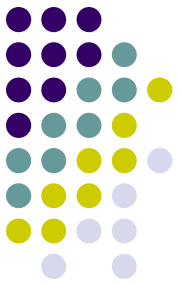
# Other Numeric Types

- Arbitrary precision
- Fraction type
- User defined types

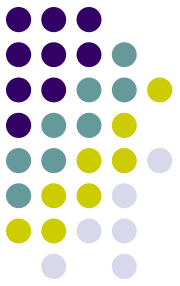


# String types

- C style strings
- C++ strings

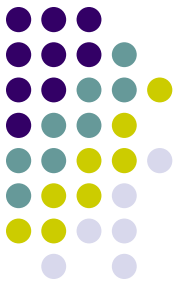


# Composite Data Types



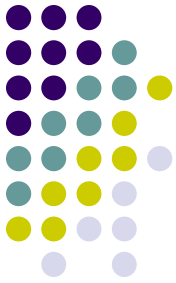
- Array
- Struct
- Class
- Union

# Contents

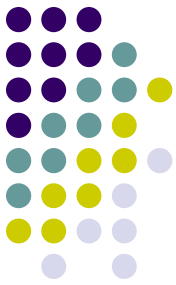


<b>Chapter 1 Preliminaries</b>	<b>25</b>
1.1 Reasons for Studying Concepts of Programming Languages.....	26
1.2 Programming Domains.....	29
1.3 Language Evaluation Criteria.....	30
1.4 Influences on Language Design.....	41
1.5 Language Categories.....	44
1.6 Language Design Trade-Offs.....	45
1.7 Implementation Methods.....	46
1.8 Programming Environments .....	53
Summary • Review Questions • Problem Set.....	54
<b>Chapter 2 Evolution of the Major Programming Languages</b>	<b>57</b>
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
<b>Chapter 3</b>	<b>Describing Syntax and Semantics</b>	<b>133</b>
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
<b>Chapter 4</b>	<b>Lexical and Syntax Analysis</b>	<b>185</b>
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
<b>Chapter 5</b>	<b>Names, Bindings, and Scopes</b>	<b>221</b>
5.1	Introduction.....	222
5.2	Names .....	223







	Contents	17
	History Note .....	223
5.3	Variables..... <a href="#">Go to page 226</a>	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
<b>Chapter 6</b>	<b>Data Types</b>	<b>259</b>
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: <b>ROBERTO IERUSALIMSCHY—Lua</b> .....	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