

Preemptive Lecture

Number Systems and Codes

A/Prof Tay Seng Chuan

1

Ground Rules (触犯者斩)

- Switch off your handphone and place it in your bag
- No talking while lecture is going on
- No gossiping while the lecture is going on
- Raise your hand if you have questions to ask
- Be on time for lecture
- Be on time to come back from the recess break to continue the lecture
- Bring your lecture notes to lecture or use the laptop.
Do not use handphone to read lecture notes.

2

Scope

Number Systems

- Positional Notations
- Decimal (base 10) Number System
- Other Number Systems & Base-R to Decimal Conversion
- Decimal-to-Binary Conversion
 - Sum-of-weights Method
 - Repeated Division-by-2 Method (for whole number)
 - Repeated Multiplication-by-2 Method (for fractions)

3

Scope (Cont.)

- Conversion between Bases
- Binary-Octal/Hexadecimal Conversion
- Binary Arithmetic Operations
- Negative Numbers
 - Sign-and-Magnitude
 - 1's Complement
 - 2's Complement
- Comparison of Sign-and-Magnitude and Complements

4

Scope (Cont.)

- Overflow
- Fixed-Point Numbers
- Floating-Point Numbers
- Arithmetic with Floating-Point Numbers
- Representing information
- Codes
 - Alphanumeric Codes
 - Error Detection Codes

5

Positional Notations

- Relative-position notation
 - Roman numerals symbols with different values: I (1), V (5), X (10), C (50), M (100)
 - Examples: I, II, III, IV, V, VI, VII, VIII, IX
 - Relative position important: IV = 4 but VI = 6
- Computations are difficult with the above the notation

6

Positional Notations (Cont.)

- Weighted-positional notation

- Decimal number system,
symbols = { 0, 1, 2, 3, ..., 9 }
- Position carries weight.
- Example:
 $(7594)_{10} = (7 \times 10^3) + (5 \times 10^2) + (9 \times 10^1) + (4 \times 10^0)$
- The value of each symbol is dependent on its type and its position in the number
- In general, $(a_n a_{n-1} \dots a_0)_{10} = (a_n \times 10^n) + (a_{n-1} \times 10^{n-1}) + \dots + (a_0 \times 10^0)$

7

Positional Notation (Cont.)

- Fractions are written in decimal numbers after the *decimal point*.

- $(2.75)_{10} = (2 \times 10^0) + (7 \times 10^{-1}) + (5 \times 10^{-2})$
- In general, $(a_n a_{n-1} \dots a_0 . f_1 f_2 \dots f_m)_{10} =$
 $(a_n \times 10^n) + (a_{n-1} \times 10^{n-1}) + \dots + (a_0 \times 10^0) +$
 $(f_1 \times 10^{-1}) + (f_2 \times 10^{-2}) + \dots + (f_m \times 10^{-m})$

8

Decimal (base 10) Number System

- Weighting factors (or weights) are in powers-of-10:
... 10^3 10^2 10^1 10^0 10^{-1} 10^{-2} 10^{-3} 10^{-4} ...
- To evaluate the decimal number 593.68, the digit in each position is multiplied by the corresponding weight:
$$5 \times 10^2 + 9 \times 10^1 + 3 \times 10^0 + 6 \times 10^{-1} + 8 \times 10^{-2}$$
$$= (593.68)_{10}$$

9

Base-R to Decimal Conversion

- **Binary** (base 2): weights in powers of 2.
– Binary digits (bits): **0, 1**.
- **Octal** (base 8): weights in powers of 8.
– Octal digits: **0, 1, 2, 3, 4, 5, 6, 7**.
- **Hexadecimal** (base 16): weights in powers of 16.
– Hexadecimal digits: **0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F**.
- **Base R**: weights in powers of R.

10



Common base used in Disneyland most likely is base 8.

11

Base-R to Decimal Conversion (Cont.)

- $(1101.101)_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-3}$
 $= 8 + 4 + 1 + 0.5 + 0.125 = (13.625)_{10}$
- $(572.6)_8 = 5 \times 8^2 + 7 \times 8^1 + 2 \times 8^0 + 6 \times 8^{-1}$
 $= 320 + 56 + 2 + 0.75 = (378.75)_{10}$
- $(2A.8)_{16} = 2 \times 16^1 + 10 \times 16^0 + 8 \times 16^{-1}$
 $= 32 + 10 + 0.5 = (42.5)_{10}$
- $(341.24)_5 = 3 \times 5^2 + 4 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} + 4 \times 5^{-2}$
 $= 75 + 20 + 1 + 0.4 + 0.16 = (96.56)_{10}$

12

Decimal-to-Binary Conversion

- Sum-of-Weights Method (You must be able to figure it out!)
- Repeated Division-by-2 Method (for whole numbers)
- Repeated Multiplication-by-2 Method (for fractions)

13

Sum-of-Weights Method

- Determine the set of binary weights whose sum is equal to the decimal number.

$$(9)_{10} = 8 + 1 = 2^3 + 2^0 = (1001)_2$$

$$(18)_{10} = 16 + 2 = 2^4 + 2^1 = (10010)_2$$

$$(58)_{10} = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 = (111010)_2$$

$$(0.625)_{10} = 0.5 + 0.125 = 2^{-1} + 2^{-3} = (0.101)_2$$


14

Repeated Division-by-2 Method

- To convert a whole number to binary, use successive division by 2 until the quotient is 0. The remainders form the answer, with the first remainder as the *least significant bit (LSB)* and the last as the *most significant bit (MSB)*.

$$(43)_{10} = (101011)_2$$

2	43		
2	21	rem 1	LSB
2	10	rem 1	
2	5	rem 0	
2	2	rem 1	
2	1	rem 0	
	0	rem 1	MSB




15

Repeated Multiplication-by-2 Method

- To convert decimal fractions to binary, repeated multiplication by 2 is used, until the fractional product is 0 (or until the desired number of decimal places is achieved). The carried digits, or *carries*, produce the answer, with the first carry as the MSB, and the last as the LSB. E.g. $(0.3125)_{10} = (0.0101)_2$. But sometimes you do not get the exact conversion.

	Carry	
0.3125×2=0.625	0	← MSB
0.625×2=1.25	1	
0.25×2=0.50	0	
0.5×2=1.00	1	← LSB



16

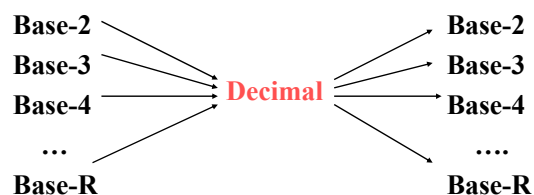
Conversion between Bases

- **Base-R to decimal**: multiply digits with their corresponding weights
- **Decimal to binary** (base 2)
 - whole numbers: repeated division-by-2
 - fractions: repeated multiplication-by-2
- **Decimal to base-R**
 - whole numbers: repeated division-by-R
 - fractions: repeated multiplication-by-R

17

Conversion between Bases (Cont.)

- In general, conversion between bases can be done via **decimal**:



Shortcuts are available for conversion between bases 2, 4, 8, 16.

18

Binary-Octal/Hexadecimal Conversion

- **Binary → Octal:** Partition in **groups of 3**
 $(10\ 111\ 011\ 001\ .\ 101\ 110)_2 = (2731.56)_8$
- **Octal → Binary:** reverse
 $(2731.56)_8 = (10\ 111\ 011\ 001\ .\ 101\ 110)_2$
- **Binary → Hexadecimal:** Partition in **groups of 4**
 $(101\ 1101\ 1001\ .\ 1011\ 1000)_2 = (5D9.B8)_{16} = 0x5D9.B8$
- **Hexadecimal → Binary:** reverse
 $(5D9.B8)_{16} = (101\ 1101\ 1001\ .\ 1011\ 1000)_2$

19

Binary Arithmetic Operations

ADDITION

- Like decimal numbers, two numbers can be added by adding each pair of digits together with carry propagation.

$(11011)_2$	$(647)_{10}$
$+ (10011)_2$	$+ (537)_{10}$
<hr/>	<hr/>
$(101110)_2$	$(1184)_{10}$
<hr/>	<hr/>

P.T.O

20

Something that we have defined!!

21

Binary Arithmetic Operations (Cont.)

$(11011)_2$	Carries	1	0	0	1	1
$+ (10011)_2$		0	1	1	0	1
<hr/>		0	1	0	0	1
$(101110)_2$		<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
		1	0	1	1	0

22

Binary Arithmetic Operations (Cont.)

SUBTRACTION

- Two numbers can be subtracted by subtracting each pair of digits together with borrowing, where needed.

$$\begin{array}{r} (11001)_2 \\ - (10011)_2 \\ \hline (00110)_2 \end{array} \qquad \begin{array}{r} (627)_{10} \\ - (537)_{10} \\ \hline (090)_{10} \end{array}$$

P.T.O

23

Binary Arithmetic Operations (Cont.)

$(11001)_2$	Borrows	0	0	-1	-1	0	0
$- (10011)_2$		0	1	1	0	0	1
$(00110)_2$		0	1	0	0	1	1
		0	0	0	1	1	0

24

Binary Arithmetic Operations (Cont.)

- MULTIPLICATION
- To multiply two numbers, take each digit of the **multiplier** and multiply it with the **multiplicand**. This produces a number of **partial products** which are then added.

$(11001)_2$	$(214)_{10}$	Multiplicand
$\times (10101)_2$	$\times (152)_{10}$	Multiplier
<hr/>	<hr/>	
$(11001)_2$	$(428)_{10}$	Partial products
$(11001)_2$	$(1070)_{10}$	
$+ (11001)_2$	$+ (214)_{10}$	
<hr/>	<hr/>	
$(1000001101)_2$	$(32528)_{10}$	Result

25

Binary Arithmetic Operations (Cont.)

DIVISION – can you figure out how this is done?

Think of the effect of shifting the bits to the left and subtract the new contents from the old contents to get the remainder.

Eg: $13_{(10)} = 1101_{(2)}$. If you shift the bits to the left by 1 place, the new contents will be $11010_{(2)} = 16 + 8 + 2$
 $= 26 = 13 \times 2$.

What will be the effect if you shift the bits to the left by 2 places, 3 places and so on?

26

Eg. $17/3 = 5.666\dots$

How is it divided in base 2?

Let's start with $6/3$.

$$\begin{array}{ccccccc}
 \begin{array}{r} 1 \\ 11 \overline{)110} \\ \underline{011} \end{array} & \rightarrow & \begin{array}{r} 10 \\ 11 \overline{)110} \\ \underline{110} \end{array} & \rightarrow & \cancel{\begin{array}{r} 100 \\ 11 \overline{)110} \\ \underline{1100} \end{array}} & \rightarrow & \begin{array}{r} 10 \\ 11 \overline{)110} \\ \underline{110} \end{array} \\
 & & & & & & \checkmark
 \end{array}$$

27

$$\begin{array}{ccccc}
 \begin{array}{r} 1 \\ 11 \overline{)10001} \\ \underline{01100} \\ \text{-----} \\ 101 \end{array} & \rightarrow & \begin{array}{r} 101 \\ 11 \overline{)10001} \\ \underline{01100} \\ \text{-----} \\ 101 \\ 11 \\ \text{-----} \\ 10 \end{array} & \rightarrow & \begin{array}{r} 101.10101 \\ 11 \overline{)10001} \\ \underline{01100} \\ \text{-----} \\ 101 \\ 11 \\ \text{-----} \\ 10 \end{array} \\
 17/3 = ?? & & & &
 \end{array}$$

$$101.10101_{(2)} = 4 + 1 + 0.5 + 0.125 + 0.03125 = 5.65625$$

But $17/3 = 5.666\dots$

$$\begin{array}{r}
 10 \text{ } 0 \\
 \underline{11} \\
 100 \\
 \underline{11} \\
 100 \\
 \underline{11} \\
 : \\
 :
 \end{array}$$

28

Negative Numbers

- Given n bits, the total number of representations is 2^n . Eg, given 4 bits , we will have $2^4 = 16$ representations.
- For the 2^n representation, *ideally*, we should allocate half of the capacity to represent positive numbers, and the other half to represent negative numbers. Eg, for 4 bits, we should be able to represent 8 positive numbers and 8 negative numbers.

29

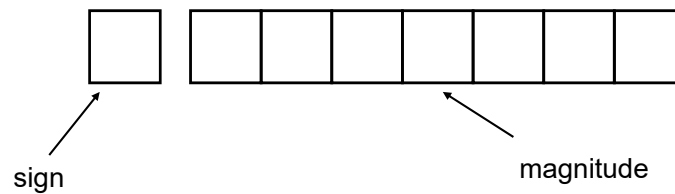
Negative Numbers: Sign-and-Magnitude

- Negative numbers are written by a minus sign in front as you have learnt in the past 17 years.
 - Example:
 $-(12)_{10}$, $-(1100)_2$
- In computer memory of fixed width, this sign is usually represented by a bit called sign bit:
 - 0 for +
 - 1 for –

30

Negative Numbers: Sign-and-Magnitude (Cont.)

- Example: an 8-bit number can have 1-bit sign and 7-bits magnitude.



31

Negative Numbers: Sign-and-Magnitude (Cont.)

8-bit Representation

- Largest Positive Number: 0 1111111 $+(127)_{10}$
- Most Negative Number: 1 1111111 $-(127)_{10}$
- Zeroes:
0 0000000 $+(0)_{10}$
1 0000000 $-(0)_{10}$
- Range for 8-bit representation: $-(127)_{10}$ to $+(127)_{10}$

32

Negative Numbers: Sign-and-Magnitude (Cont.)

- To negate a number, just invert the sign bit.
- Examples:
 - $(0\ 0100001)_{sm} \rightarrow (1\ 0100001)_{sm}$
 - $(1\ 0000101)_{sm} \rightarrow (0\ 0000101)_{sm}$

Sign-and-Magnitude representation is simple

but is not in use in any computer. **Why?** $2^8 = 256$!!

33

Signed Numbers in Binary

- Two other ways of representing signed numbers for binary numbers are:
 - 1's-complement
 - 2's-complement

34

1's Complement

- Given a number x which can be expressed as an n -bit binary number, its negative value can be obtained in 1's-complement representation using:

$$2^n - x - 1 \text{ where } 0 \leq x \leq 2^{n-1} - 1$$

Example: With an 8-bit number 00001100, its negative value, expressed in 1's complement, is represented as follows:

$$\begin{aligned} -(00001100)_2 &= -(12)_{10} \\ &= (2^8 - 12 - 1)_{10} = (256 - 12 - 1)_{10} \\ &= (243)_{10} \\ &= (11110011)_{1's} \end{aligned}$$

35

Most Negative Number Represented by 1's Complement Notation

Given a number x which can be expressed as an n -bit binary number, its negative value can be obtained in 1's-complement representation using:

$$2^n - x - 1 \text{ where } 0 \leq x \leq 2^{n-1} - 1$$

To get the most negative number, we will have to give the value x with the largest possible value. If we have only 8 bits, $n = 8$. The representation of the most negative value will be derived as follows:

$$\begin{aligned} 2^8 - (2^7 - 1) - 1 &= 256 - (128 - 1) - 1 \\ &= 128_{(10)} \rightarrow (10000000)_2 \end{aligned}$$

So the most negative number in the 1's complement notation is represented by 10000000_(1's) and it has the decimal value of -127₍₁₀₎.

36

1's Complement (Cont.)

$$-(00001100)_2 = (11110011)_{1's}$$

Given a number x which can be expressed as an n -bit binary number, its negative value can be obtained in 1's-complement representation using:

$$2^n - x - 1 \text{ where } 0 \leq x \leq 2^{n-1} - 1$$

- Shortcut: invert all the bits.
Examples: $-(00000001)_2 \rightarrow (11111110)_{1's}$
 $-(01111111)_2 \rightarrow (10000000)_{1's}$
- Largest Positive Number: $0\ 1111111 \rightarrow +(127)_{10}$
- Most Negative Number: $1\ 0000000 \rightarrow -(127)_{10}$
 $(0000000_{(2)} + 1) - 2^7 = 1 - 128 = -127$
- Zeroes: $0\ 0000000$
 $1\ 1111111$
- Range: $-(127)_{10}$ to $+(127)_{10}$
- The most significant bit still represents the sign: 0 = +ve; 1 = -ve.

37

2's Complement

- Given a number x which can be expressed as an n -bit binary number, its negative number can be obtained in 2's-complement representation using:

$$2^n - x \text{ where } 1 \leq x \leq 2^{n-1}$$

Example: With an 8-bit number 00001100, its negative value in 2's complement is represented as follows:

$$\begin{aligned} -(00001100)_2 &\rightarrow -(12)_{10} \\ &\rightarrow (2^8 - 12)_{10} = (256 - 12)_{10} \\ &= (244)_{10} \\ &\rightarrow (11110100)_{2's} \end{aligned}$$

38

2's Complement (Cont.)

$$-(00001100)_2 = (11110100)_{2's}$$

- Shortcut: invert all the bits and add 1.

Examples: $-1_{10} = (??)_{2's}$

$$\begin{aligned} -(00000001)_2 &\rightarrow (11111110)_{1's} \quad (\text{invert}) \\ &\rightarrow (11111111)_{2's} \quad (\text{add 1}) \end{aligned}$$

$$-126_{10} = (??)_{2's}$$

$$\begin{aligned} -(01111110)_2 &\rightarrow (10000001)_{1's} \quad (\text{invert}) \\ &\rightarrow (10000010)_{2's} \quad (\text{add 1}) \end{aligned}$$

39

Most Negative Number Represented by 2's Complement Notation

Given a number x which can be expressed as an n -bit binary number, its negative value can be obtained in 2's-complement representation using:

$$2^n - x \quad \text{where } 1 \leq x \leq 2^{n-1}$$

To get the most negative number, we will have to give the value x with the largest possible value. If we have only 8 bits, $n = 8$. The representation of the most negative value will be derived as follows:

$$\begin{aligned} 2^8 - 2^{8-1} &= 2^8 - 2^7 = 256 - 128 \\ &= 128_{(10)} \rightarrow (10000000)_2 \end{aligned}$$

So the most negative number in the 2's complement notation is represented by $10000000_{(2's)}$ and it has the decimal value of $-128_{(10)}$.

40

2's Complement (Cont.)

Given a number x which can be expressed as an n -bit binary number, its negative number can be obtained in 2's-complement representation using:

$$2^n - x \text{ where } 1 \leq x \leq 2^{n-1}$$

- Largest Positive Number: 0 111111 $+(127)_{10}$
- Most Negative Number: 1 000000 $-(128)_{10}$
 $000000_{(2)} - 2^7 = 0 - 128 = -128$
- Zero: 0 0000000
- Range: $-(128)_{10}$ to $+(127)_{10}$
- The most significant bit still represents the sign: 0 = +ve; 1 = -ve.

41

Comparisons of Sign-and-Magnitude & Complements

Example: 4-bit signed number

Value	Sign-and-Magnitude	1's Comp.	2's Comp.	Value	Sign-and-Magnitude	1's Comp.	2's Comp.
+7	0111	0111	0111	-0	1000	1111	-
+6	0110	0110	0110	-1	1001	1110	1111
+5	0101	0101	0101	-2	1010	1101	1110
+4	0100	0100	0100	-3	1011	1100	1101
+3	0011	0011	0011	-4	1100	1011	1100
+2	0010	0010	0010	-5	1101	1010	1011
+1	0001	0001	0001	-6	1110	1001	1010
+0	0000	0000	0000	-7	1111	1000	1001
				-8	-	-	1000

42

Overflow Problems

- Signed binary numbers are of a fixed range.
- If the result of addition/subtraction goes beyond this range, overflow occurs.
- Two conditions under which overflow can occur are:
 - (i) *positive add positive gives negative*
 - (ii) *negative add negative gives positive*

43

Overflow Problem (Cont.)

- Examples: 4-bit numbers (in 2's complement)
- Range : $(1000)_{2's}$ to $(0111)_{2's}$ or $(-8_{10}$ to $7_{10})$
 - (i) $(0101)_{2's} + (0110)_{2's} = (1011)_{2's}$
 $(5)_{10} + (6)_{10} = -(5)_{10} ?!$ (overflow!)
 - (ii) $(1001)_{2's} + (1101)_{2's} = (\underline{1}0110)_{2's}$ (discard end-carry)
 $= (0110)_{2's}$
 $(-7)_{10} + (-3)_{10} = (6)_{10} ?!$ (overflow!)

44

Floating Point Numbers

I will only give an overview for this bridging course.

The number of real numbers is infinite and real numbers are uncountable.

Floating number notation is used to represent a very small subset of real numbers. The number of floating numbers is finite and floating numbers are countable.

45

Floating Point Numbers

- Fixed point numbers have limited range.
- To represent very large or very small numbers, we use floating point numbers (cf. scientific numbers). Examples:
 - 0.23×10^{23} (very large positive number)
 - -0.1239×10^{-10} (very small negative number)

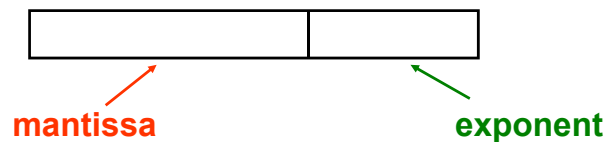
46

Floating Point Numbers (Cont.)

- Floating point numbers have three parts:

0.52487×10^{42}
mantissa, base, and exponent

- Base is usually fixed for each number system.
- Therefore, we only need to represent *mantissa* and *exponent*.



47

Floating Point Numbers (Cont.)

- Mantissa is usually in normalised form
($0.y_1y_2y_3 \times 10^z$, where $y_1 > 0$, in this case):
 - (base 10) $23_{(10)} \times 10^{21}$ normalised to 0.23×10^{23}
 - (base 10) $-0.0017_{(10)} \times 10^{21}$ normalised to -0.17×10^{19}
 - (base 2) $0.01101_{(2)} \times 2^3$ normalised to 0.1101×2^2
- A 16-bit floating point number may have 10-bit mantissa and 6-bit exponent.
- More bits in exponent gives larger range.
- More bits for mantissa gives better precision.
- The real number 0 does **not** have any representation in this form. (To be explained later!)
- The number of real numbers represented in floating point is countable and is finite.

48

(base 2) $0.01101_{(2)} \times 2^3$ normalised to 0.1101×2^2

$$\begin{aligned} 0.01101_{(2)} \times 2^3 &= 0.01101_{(2)} \times 2/2 \times 2^3 \\ &= (0.01101_{(2)} \times 2) \times (2^3 / 2) \\ &= 0.1101_{(2)} \times 2^2 \end{aligned}$$

normalisation

49

Floating Point (Detail)

You can write any number N in the following format :

$$N = f \times b^e$$

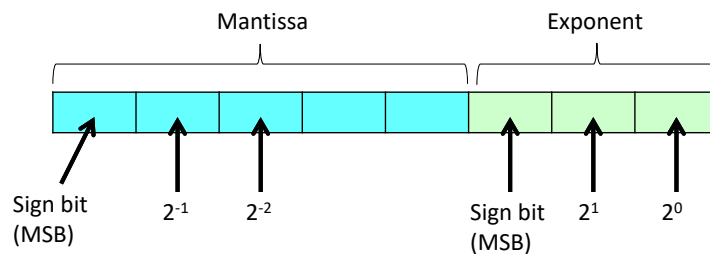
where : f is a fraction called the mantissa

b is the radix (or base), and

e is the exponent, which is an integer

Mantissa and exponent are represented by the 2's complement notation.

If given 8 bits where 5 bits are mantissa, we will have only 3 bits for exponent as follow :



50

For example: **10011** **011** $\rightarrow 1+2=3$

$10011 \rightarrow 01100 + 1 = 01101$

$$= -(0.5 + 0.25 + 0 + 0.0625) = -0.8125_{(10)}$$

Or, $10011 \rightarrow 0 + 0 + 2^{-3} + 2^{-4} - 2^0 = 0.125 + 0.0625 - 1 = -0.8125_{(10)}$

Thus **10011 011** has the decimal value of

$$-0.8125_{(10)} \times 2^3 = 0.8125_{(10)} \times 8_{(10)} = -6.5_{(10)}$$

The format imposes that the leftmost 2 bits of mantissa must have different values, ie, 01....., or 10.....

If the mantissa is positive, its range will be $0.5 \leq f < 1$

ie, 01000 (0.5), to, 01111 ($0.5 + 0.25 + 0.125 + 0.0625$)
 < 1

If the mantissa is negative, its range will be $-1 \leq f < -0.5$

ie 10000 (0-1=-1), to, 10111 ($0.25+0.125+0.0625 - 1 < -0.5$)
 < -0.5

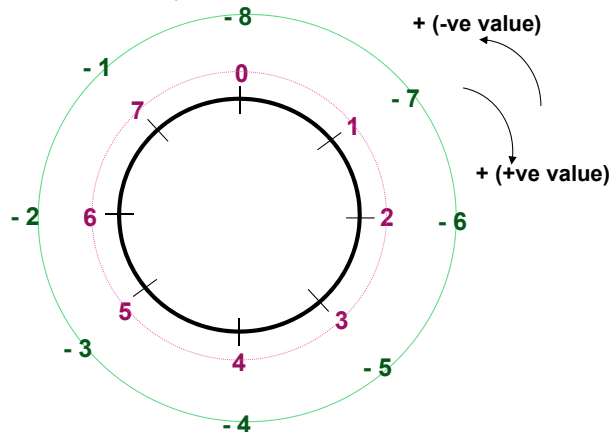
$$\begin{aligned} (< 0.5) - 1 &= (< 0.5) - 0.5 - 0.5 \\ &= ((< 0.5) - 0.5) - 0.5 \\ &= (\text{smaller} - \text{bigger}) - 0.5 \\ &= (\text{negative value}) + (-0.5) \\ &< -0.5 \end{aligned}$$

51

The Cyclic Behaviors of 2's Complement Notation (-8 to 7)

Value	2's Comp.
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000

Value	2's Comp.
-0	-
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

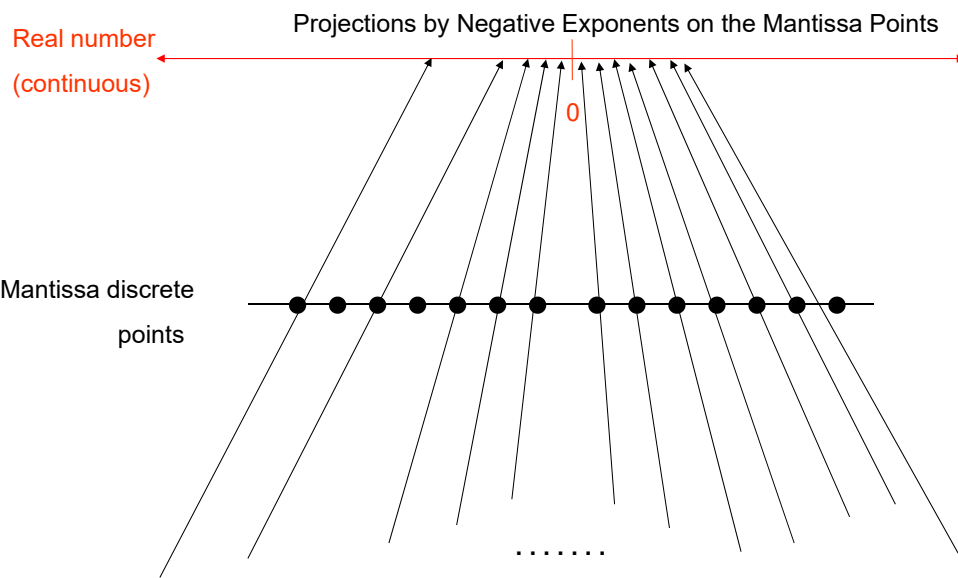


$$6_{(10)} + 7_{(10)} = 0110_{(2's)} + 0111_{(2's)} = 1101_{(2's)} = -3_{(10)}$$

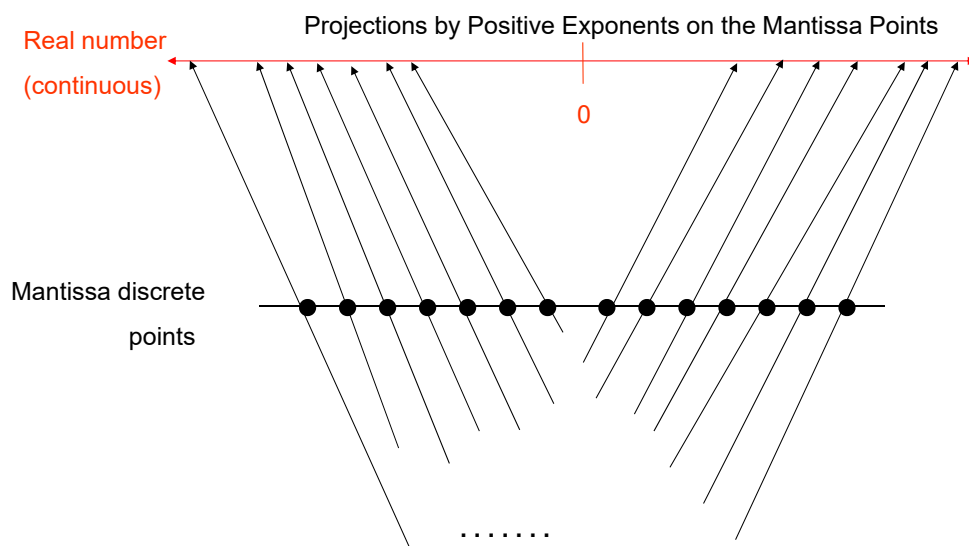
$$4_{(10)} + 6_{(10)} = 0100_{(2's)} + 0110_{(2's)} = 1010_{(2's)} = -6_{(10)}$$

$$-4_{(10)} + (-7)_{(10)} = 1100_{(2's)} + 1001_{(2's)} = 0101_{(2's)} = 5_{(10)}$$

52



53



54

Example:

Covert $-0.377_{(10)}$ to notation with 5 mantissa bits and 3 exponent bits.

$$-0.377_{(10)} = -0.377_{(10)} \times 2^0 \text{ but } -0.377_{(10)} \text{ is not smaller than } -0.5.$$

$$(-1 \leq f < -0.5)$$

$$-0.377_{(10)} = -0.754_{(10)} \times 2^{-1} \text{ (times 2 and divides 2). It is OK to proceed now.}$$

We will work on $0.754_{(10)}$ by repeated multiplication by 2.

	<u>Integral Value</u>
$0.754 \times 2 = 1.508$	1
$0.508 \times 2 = 1.016$	1
$0.016 \times 2 = 0.032$	0
$0.032 \times 2 = 0.064$	0
$0.064 \times 2 = 0.128$	0
$0.128 \times 2 = 0.256$	0



$$\text{So, } -0.377_{(10)} = -0.110000..._{(2)} \times 2^{-1}$$

55

$$-0.377_{(10)} = \underbrace{-0.110000...}_{(2)} \times 2^{-1}$$

Mantissa has 5 bits

$$\begin{array}{r} -0.1100_{(2)} \\ \rightarrow 1\ 0011_{(1's)} \\ \quad 1\ (+) \\ \hline 1\ 0100_{(2's)} \end{array}$$

Exponent has 3 bits

$$\begin{array}{r} -1 = -001_{(2)} \\ \downarrow \\ 110_{(1's)} \\ \quad 1\ (+) \\ \hline 111_{(2's)} \end{array}$$

$-0.377_{(10)}$ is represented by

1	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---

56

But something is not quite right for 10100111

Mantissa

$$\underline{10100}_{(2's)} \rightarrow 0.25_{(10)} - 1_{(10)} = -0.75_{(10)}$$

Exponent

$$\underline{111}_{(2's)} \rightarrow (1+2)_{(10)} - 4_{(10)} = -1_{(10)}$$

This gives $-0.75 \times 2^{-1} = -0.375_{(10)}$

This is **not** equal to -0.377.

This kind of error is called truncation error.

57

Next, can you represent the real number 0 by the mantissa and exponent notation ($f \times 2^e$)? (We will explain later)

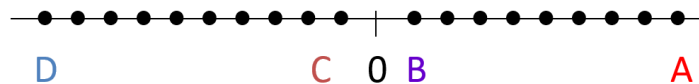
For 5 mantissa bits and 3 exponent bits, what are

the largest positive number (A)?

the smallest positive number (B)?

the least negative number (C)?

the most negative number (D)?



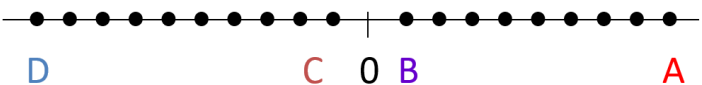
For exponent e with 3 bits, its range is -4 (100) to 3 (011).

Largest positive mantissa f : $\underline{0}1111 = 0.5 + 0.25 + 0.125 + 0.0625$
 $= 0.9375_{(10)}$

Smallest positive mantissa f : $\underline{0}1000 = 0.5_{(10)}$

$$\rightarrow \begin{aligned} A &= 0.9375 \times 2^3 = 0.9375 \times 8 = 7.5_{(10)} \\ B &= 0.5 \times 2^{-4} = 0.5 \times 0.0625 = 0.03125_{(10)} \end{aligned}$$

58



Least negative f 10111 \rightarrow -[01000

$$\begin{array}{r} 1 (+)] \\ \hline - 01001 \end{array}$$

- (0.5 + 0.0625) = -0.5625

$$C = -0.5625 \times 2^{-4}$$

$$= -0.5625 \times 0.0625 = -0.03515625_{(10)}$$

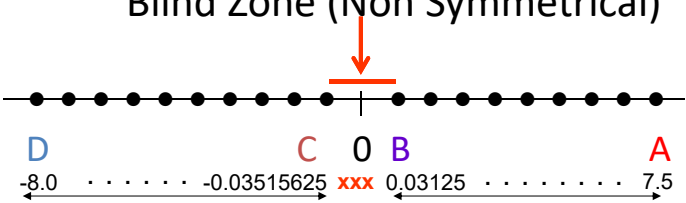
Most negative f 10000 = -1₍₁₀₎

$$D = -1 \times 2^3 = -1 \times 8$$

$$= -8.0_{(10)}$$

59

Blind Zone (Non Symmetrical)



8 bits \rightarrow 2^7 floating points
 16 bits \rightarrow 2^{15} floating points
 Because for 2 bits we have

00	}	4 cases
01		
10		
11		

Answer: The real number 0 is not represented in the mantissa and exponent notation.

But the mantissa includes only 01..... and 10.....
 due to normalization!

$\underbrace{\hspace{2cm}}$
 Positive number

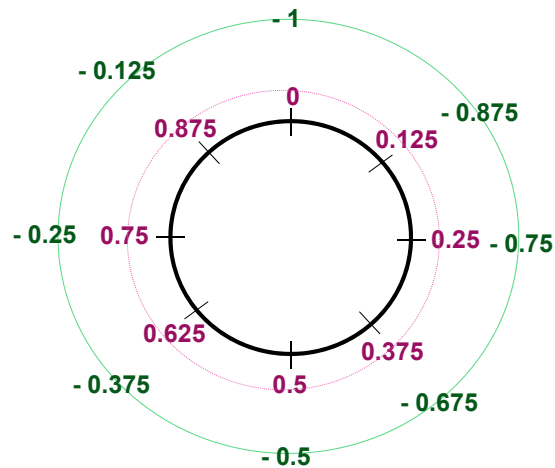
$\underbrace{\hspace{2cm}}$
 negative numbers

60

The Cyclic Behaviors of 2's Complement Notation for Mantissa with 4 bits (-1 to 0.875)

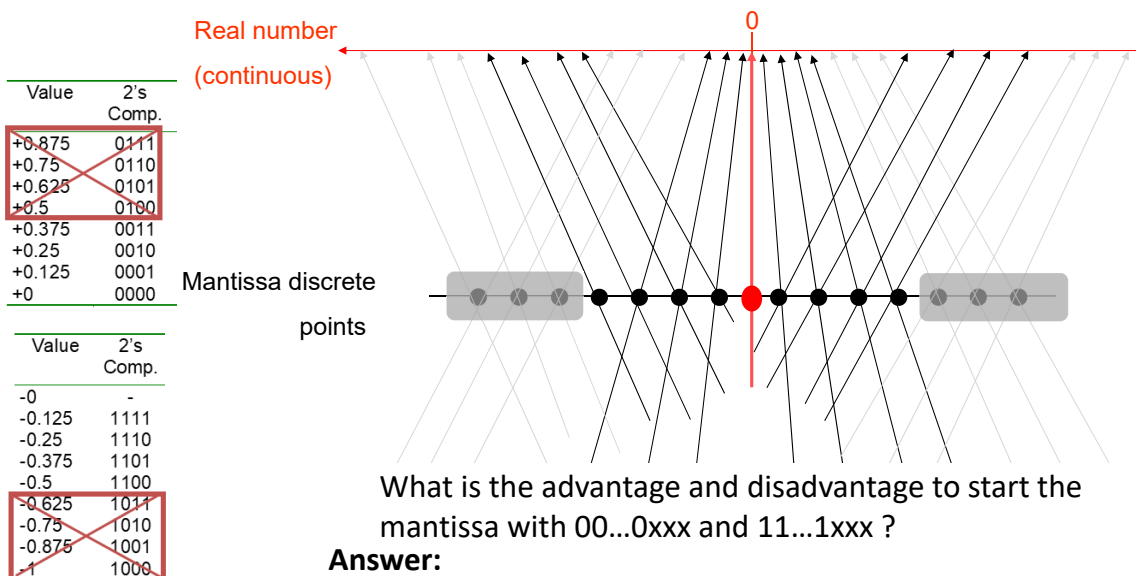
Value	2's Comp.
+0.875	0111
+0.75	0110
+0.625	0101
+0.5	0100
+0.375	0011
+0.25	0010
+0.125	0001
+0	0000

Value	2's Comp.
-0	-
-0.125	1111
-0.25	1110
-0.375	1101
-0.5	1100
-0.625	1011
-0.75	1010
-0.875	1001
-1	1000



What will be the advantage and disadvantage if we were to start the mantissa with 00...0xxx and 11...1xxx ?

61



What is the advantage and disadvantage to start the mantissa with 00...0xxx and 11...1xxx ?

Answer:

The advantage is that the real number 0 can be represented as a floating-point number, and the disadvantage is that the range of the representations will be shortened at both ends.

62

What have you learnt?

- There is no perfect solution when the total capacity is finite.
- Similarly, you cannot have the best of both worlds if you have only limited resources. Eg, you have only 24 hours in a day. So you must learn to choose. You cannot have every thing. This is the law of nature!

63

Given only 16 bits

- What will be the effect if you have more exponent bits than mantissa bits?

Answer: The range of the floating numbers is expanded, but the precision is degraded.

- What will be the effect if you have more mantissa bits than exponent bits?

Answer: The precision of the floating numbers is better as the points are closer to the actual values, but the range will not be wide.

You cannot have both the fish and the bear's paw!!

64

Arithmetic with Floating Point Numbers

- Arithmetic is more difficult for floating point numbers.

- MULTIPLICATION

Steps: (i) multiply the mantissa
(ii) add-up the exponents
(iii) normalise

- **Example:**

$$\begin{aligned} & (0.12 \times 10^2)_{10} \times (0.2 \times 10^{30})_{10} \\ &= (0.12 \times 0.2)_{10} \times 10^{2+30} \\ &= (0.024)_{10} \times 10^{32} \quad (\text{normalise}) \\ &= (0.24 \times 10^{31})_{10} \end{aligned}$$

65

Arithmetic with Floating Point Numbers (Cont.)

ADDITION

Steps: (i) equalise the exponents
(ii) add-up the mantissa
(iii) normalise

- **Example:**

$$\begin{aligned} & (0.12 \times 10^3)_{10} + (0.2 \times 10^2)_{10} \\ &= (0.12 \times 10^3)_{10} + (0.02 \times 10^3)_{10} \quad (\text{equalise exponents}) \\ &= (0.12 + 0.02)_{10} \times 10^3 \quad (\text{add mantissa}) \\ &= (0.14 \times 10^3)_{10} \end{aligned}$$

- **Can you figure out how to perform SUBTRACTION and DIVISION for (binary/decimal) floating-point numbers?**

66

Alphanumeric Codes

- Apart from numbers, computers also handle textual data.
- Character set frequently used includes:
 - alphabets: 'A' .. 'Z', and 'a' .. 'z'
 - digits: '0' .. '9'
 - special symbols: '\$', ':', ';', '@', '*', ...
 - non-printable: SOH, NULL, BELL, ...
- Usually, these characters can be represented using 7 or 8 bits.

67

Alphanumeric Codes (Cont.)

Two widely used standards:

ASCII (American Standard Code for Information Interchange)

EBCDIC (Extended BCD Interchange Code)

- **ASCII**: 7-bit, plus a *parity bit* for error detection (odd/even parity).
- **EBCDIC**: 8-bit code.

Character	ASCII Code
0	0110000
1	0110001
...	...
9	0111001
:	0111010
A	1000001
B	1000010
...	...
Z	1011010
[1011011
\	1011100

68

Alphanumeric Codes (Cont.)

ASCII table:

LSBs	MSBs							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC ₁	!	1	A	Q	a	q
0010	STX	DC ₂	"	2	B	R	b	r
0011	ETX	DC ₃	#	3	C	S	c	s
0100	EOT	DC ₄	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	O	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

69

Error Detection Codes

- Errors can occur during data transmission. They should be detected, so that re-transmission can be requested.
- With binary numbers, usually single-bit errors occur.
Example: 0010 erroneously transmitted as 0011, or 0000, or 0110, or 1010.
- For single-error detection, one additional bit is needed.

70

Error Detection Codes (Cont.)

- Parity bit.
 - Even parity: additional bit supplied to make total number of '1's even.
 - Odd parity: additional bit supplied to make total number of '1's odd.

- Example: Odd parity.

Character	ASCII Code
0	0110000 1
1	0110001 0
...	...
9	0111001 1
:	0111010 1
A	1000001 1
B	1000010 1
...	...
Z	1011010 1
[1011011 0
\	1011100 1

Parity bits

71

Error Detection Codes (Cont.)

- Parity bit can detect odd number of errors but not even number of errors.

Example: For odd parity numbers,

10011 → 10001 (detected)

10011 → 10101 (not detected!)

- Parity bits can also be applied to a block of data:

Row-wise parity

0110	1
0001	0
1011	0
1111	1
1001	1
0101	0

Column-wise parity

- Sometimes, it is not enough to do error detection. We may want to do error correction. Error correction can be done but is more expensive. You can study Mathematics in NUS to know error correction codes.

72

Big Idea: Type not associated with Data

0011 0100 0101 0101 0100 0011 0100 0010

- What does bit pattern mean:
 - 2.034×10^{-4} ? 878,003,010? “4UCB”?
 - Data can be anything; operation that uses the data determines its type!

73

Summary

Number Systems

- Positional Notations
- Base conversion
- Negative Numbers
 - Sign-and-Magnitude
 - Complement
 - Subtraction
- Floating Point
- Codes
 - Alphanumeric Codes
- Big idea
 - Data can be anything; operation that uses the data determines its type!

74

We are really for C Programming now!

75

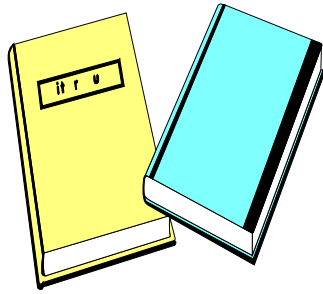
Ground Rules (触犯者斩, 真的斩)

- Switch off your handphone and pager
- No talking while lecture is going on
- No gossiping while the lecture is going on
- Raise your hand if you have question to ask
- Be on time for lecture
- Be on time to come back from the recess break to continue the lecture
- Bring your printed lecture notes to lecture or use a laptop. Do not use handphone.

Objective of C Programming

- Aims to build a strong foundation for programming skills and therefore the training shall start from the very basic level.
- Visual C++ is used in computer labs.
- Three different activities: lecture, tutorial (6 sets) and practical (13 sets). You have to dirty your hands to really learn programming.

76

**Text Book:**

Learning C

Neil Graham, McGraw-Hill
International Editions

ISBN 0-07-112628-7

Reference:

Application Programming in Ansi C
3rd Edition

Richard Johnsonbaugh and Martin Kalin

ISBN 0-13-394222-8

There are many books on C programming language in the libraries.

77

Contents

1. Getting Started
2. Control Statements and
Related Operators
3. Input and Output
4. Arrays
5. Pointers and Arrays
7. Types and Conversions
8. More on Macros

78

Chapter 1 Getting Started

Ground Rules

- Switch off your handphone and pager
- Switch off your laptop computer and keep it
- No talking while lecture is going on
- No gossiping while the lecture is going on
- Raise your hand if you have question to ask
- Be on time for lecture
- Be on time to come back from the recess break to continue the lecture
- Bring your lecture notes to lecture

1.1 Functions

- A C program is a collection of *functions or named control blocks*. What is a program ?
- How does a function look like ?
- What have you learnt in your High School in China about a mathematical function?
 - function is defined on a domain, which can be treated as the values *input to the function* (something that you feed into function).
 - function has a range, which can be considered as the values output by the function (something the function gives you at the end of its execution).

79

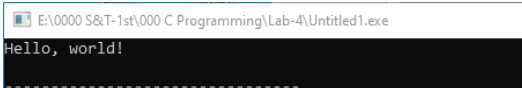
- What is “at the end of its execution” ?
- The function main() is the initial control block of a C program. The program execution starts from the function main().

1.2 The Hello-World Program

L1-1.c

```
/* File hello.c */
/* Say hello to user */
#include <stdio.h>
int main (void)
{
    printf("Hello, world!\n");
    return 0;
}
```

Screen Output :



The above program can be written as

```
#include <stdio.h>
void main ()
{
    printf("Hello, world!\n");
}
```

In Turbo C, the program can also be written as

```
#include <stdio.h>
main ()
{
    printf("Hello, world!\n");
    return 0;
}
```

80

1.3 Function Definition

A function definition has the following form:

```
heading
{
    declarations (if any)

    statements
}
```

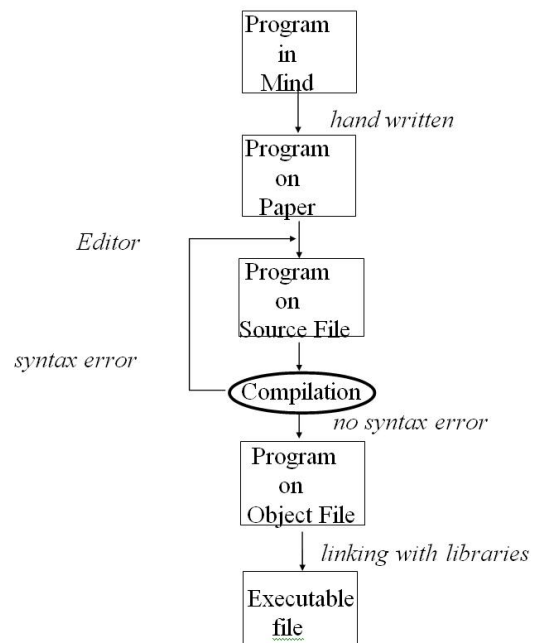
Format of Heading

Function_Type *Function_Heading* (optional input)

The main() function has a default **int** type.

81

1.4 Editing, Compiling, and Linking



82

1.5 Identifiers (function names, constants, variables)

Names, or identifiers, must be formed according to certain rules:

- An identifier can contain only letters, digits, and the underscore character `_`.
- An identifier must begin with a letter of the alphabet or an underscore character.
- Identifiers that begin with an underscore are reserved for the implementation and should not be defined by the programmer.
- An identifier must not be the same as one of the keywords listed in Appendix 1 of text book (page 283).
- C distinguishes between uppercase and lowercase letters (Sum and sum are not the same).
- Only the first 31 characters of an identifier are significant.

*An identifier **cannot** have more than one type.*

An identifier defined by the programmer should not be the same as one already defined in the library.

Exercise: Which of the following are legal identifiers created by programmer ?

2sum .main main sum2

s2sum this1 AbCdeFgh

The *scope* of an identifier is that part of the program in which the identifier can be used to access its object.

Two types of scope :

local (confined) and global (full, any where).

83



auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while
enum	signed	

84

1.6 Declaring, Defining, and Calling Functions

Listing L1-2.c shows a slightly more elaborate greeting program that defines three functions: `main()`, `say_hello()`, and `say_goodbye()`. This is for teaching purpose. You can certainly write it differently.

Screen Output:

```
E:\0000 S&T-1st\000 C Programming\Lab-4\Untitled1.exe
Hello, everybody!
Best wishes to all!
I've got to go now.
So long, and have a nice day!
-----
Process exited after 11.59 seconds with return value 0
Press any key to continue . . . _
```

L1-2.c

```
/* File greet.c */
/* Illustrates function definitions and declarations */
#include <stdio.h>
void say_hello(void);
void say_goodbye(void);

int main (void)
{
    say_hello();
    say_goodbye();
    return 0;
}

/* Extend greetings and best wishes */
void say_hello(void)
{
    printf("Hello, everybody!\n");
    printf("Best wishes to all!\n");
}

void say_goodbye(void)
{
    printf("I've got to go now.\n");
    printf("So long, and have a nice day!\n");
}
```

85

1.7 Types and Values

- C classifies data values as belonging to different data types.
- The type of a value determines how it is stored in memory and what operations can be carried out on it. E.g., the letter 'A' does not have square root.

Arithmetic types

- integer types for whole numbers
e.g. `int i, j;`
- floating types for floating-point numbers (real numbers)
e.g. `float p, q;`

86

1.8 Variables and Assignment

- A variable is a memory location named by an identifier.
- The value stored in the memory location is the current value of the variable, and can be changed by an assignment operation, which stores a new value in the memory location.
- A variable must always be declared before it is used. The declaration must specify the data type of the value of the variable.

For example,

```
int i, j, k;          /* variable declarations
    int has 4 bytes in DEV C++ and the range is form -2,147,483,648 to 2,147,483,647 */
float sum, initial;

i = 30;
j = 40;
k = 50;
sum = 0;
initial = 30.2;
```

87

Declaring a variable with initial value

```
int i = 30, j = 40, k = 50;
float sum = 0, initial = 30.2;
```

1.9 Arithmetic Operators

Integer Types		Floating Types	
+	addition	+	addition
-	subtraction	-	subtraction
*	multiplication	*	multiplication
/	integer division: quotient	/	floating-point division
%	integer division: remainder		
Examples: 9+2 = 11 7-9 = -2 9%4 = 1 11/2 = 5		Examples: 9+2.0 = 11.0 6*2.0 = 12.0 11/2.0 = 5.5	

88

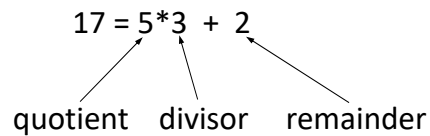
Declaring a variable with initial value

int i = 30, j = 40, k = 50;

float sum = 0, initial = 30.2;

1.9 Arithmetic Operators

Eg: 17/3

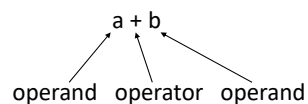


(In C language, the remainder follows the sign of numerator, and the magnitude is equal to abs(numer) % abs(denom))

89

1.10 Expression Evaluation

- Operators are grouped with operands in order of decreasing precedence. E.g., * and / are performed before + and -. The precedence table is given in page 285 of text book.
- The highest-precedence operators are grouped with their operands, then the operators with next highest precedence, and so on.



Suppose a = 6, b = 2, c = 9 (assume integer type). What are the values of the following expressions ?

a-b*c

a/b-c+a/c

a/(b-c+a)/c

a*b+(c+b*c)*a

90

1.11 Side Effects

- Value and side effects of an expression depend on the order of assignments and operations. This order can sometime cause confusion.
- Use brackets for clarity.

1.12 Standard Input and Output

- The stdio (read as “standard I O”) library provides a number of input and output functions. The most useful functions are scanf() and printf().

```
printf("%d plus %d equals %d\n", 20, 30, 50);
```

%f for float-point number (real number)

%c for character

91

- The arguments of printf(), like all function arguments, can be represented by variables or expressions. For example, the statements

```
int a=20, b=30;
```

```
printf("%d plus %d equals %d\n", a, b, a+b);
```

produce the same output as the single printf() statement above.

What are the arguments of a function ?

92

1.13 Field Widths, Justification, and Precision

- By default, a printed value is right justified in its field. What is by default ?
- A minus sign preceding the field-width causes the printed value to be left-justified.
- The field width can be followed by a decimal point and a precision.

Examples:

```
printf("%-10s %-7s\n\n", "Item", "Price");  
printf("%-10s %-7.2f\n", "Computer", 1999.95);  
printf("%-10s %-7.2f\n", "Typewriter", 249.98);  
printf("%-10s %-7.2f\n", "Paper", 5.99);
```

I	t	e	m							P	r	i	c	e		
C	o	m	p	u	t	e	r			1	9	9	9	.	9	5
T	y	p	e	w	r	i	t	e	r	2	4	9	.	9	8	
P	a	p	e	r						5	.	9	9			

produce the following printout:

93

1.14 Input with scanf()

scanf() is similar to the output function printf()

- input is controlled by a formatted string
- conversion specifier for type float is **f**, double is **lf**. What is a float and what is a double ?
- major difference between scanf() and printf():
printf(): passing by value
scanf() : passing by address (or passing by reference) &

address refers to the House Number!

What is & (ampersand) ?

94

L1-3.c

```
/* File rectangl.c */
/* Compute area and perimeter of rectangle */
#include <stdio.h>
int main (void)
{
    float length, width; /* Data */
    float area, perimeter; /* Results */

    printf("Enter length: ");
    scanf("%f", &length);
    printf("Enter width: ");
    scanf("%f", &width);

    area = length * width;
    perimeter = 2 * (length + width);
    printf("Area = %.3f\n", area);
    printf("Perimeter = %.3f\n", perimeter);
    return 0;
}
```

Screen Output:

E:\0000 S&T-1st\000 C Programming\Lab-4\Untitled1.exe

```
Enter length: 8.2
Enter width: 2.44
Area = 20.008
Perimeter = 21.280

-----
Process exited after 45.61 seconds with return val
Press any key to continue . . .
```

95

1.15 Named Constants and Preprocessor Macros

- define names for numerical constants and use the names rather than the numbers throughout the program
- the name of a constant should reflect the context of its use
- named constant saves program maintenance time
- named constant is also called macro

A macro is an identifier defined, using **#define** directive, to represent a segment of program text. After the definition has been processed, the preprocessor locates each occurrence of the identifier and expands it by replacing it with the corresponding text.

It is customary in C to use all capital letters for identifiers that represent constants.

What if I purposely use small letters for predefined constant ?

96

96

L1-4.c

```

/* Program to make change */
#include <stdio.h>
/* Define values of coins */
#define HALF      50
#define QUARTER   25
#define DIME      10
#define NICKEL    5
int main (void)
{
    int change; /* Amount of change yet to be returned */
    int coins;  /* Number of coins to be returned for a
                particular denomination */
    printf("Enter amount of change in cents: ");
    scanf("%d", &change);
    coins = change / HALF;
    change = change % HALF;
    printf("%-9s %2d\n", "Halves:", coins);
    coins = change / QUARTER;
    change = change % QUARTER;
    printf("%-9s %2d\n", "Quarters:", coins);
    coins = change / DIME;
    change = change % DIME;
    printf("%-9s %2d\n", "Dimes:", coins);
    coins = change / NICKEL;
    change = change % NICKEL;
    printf("%-9s %2d\n", "Nickels:", coins);
    printf("%-9s %2d\n", "Pennies:", change);
    return 0;
}

```

Screen Output:

E:\0000 S&T-1st\000 C Programming\Lab-4\Untitled1.exe

```

Enter amount of change in cents: 97
Halves:      1
Quarters:    1
Dimes:       2
Nickels:     0
Pennies:     2

-----
Process exited after 21.31 seconds with return value 0
Press any key to continue . . .

```

97

97

L1-5.c

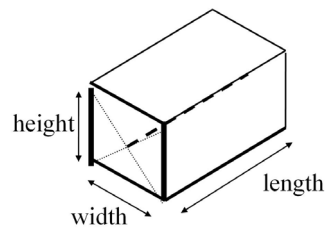
1.16 Functions with Arguments

```

/* Demonstrate declaring and defining function
   having arguments and return value */
#include <stdio.h>
float size(float, float, float);
int main (void)
{
    float len, wdth, hght, sz;
    printf("Enter length, width, and height: ");
    scanf("%f%f%f", &len, &wdth, &hght);
    sz = size(len, wdth, hght); /* actual arguments */
    printf("Size is %.2f\n", sz);
    return 0;
}

/* Compute size of rectangular box */
float size(float length, float width, float height)
{
    /* formal arguments */
    float size1, girth;
    girth = 2 * (width + height);
    size1 = length + girth;
    return size1;
}

```



Screen Output:

E:\0000 S&T-1st\000 C Programming\Lab-4\Untitled1.exe

```

Enter length, width, and height: 8.0 5.0 3.0
Size is 24.00

-----
Process exited after 25.24 seconds with return value 0
Press any key to continue . . .

```

The flow of data is strictly one-way: the function cannot change the values of the arguments passed to the function. In particular, if the function assigns a new value to an argument, only the value of the argument in the function is changed. The corresponding argument from the caller will still keep the original value.

98