



57

Outline

- **(Primitive) Types and sizes**
 - **Types:** char, short, int, long, unsigned short, unsigned int, float, double
 - **Constant values (literals)**
 - char
 - int
 - float
- (Structured) Array and “strings”
- Expressions
 - Basic operators
 - Type promotion and conversion
 - Other operators
 - Precedence of operators

Plan for today

58

YORK
UNIVERSITY

58

Character Constants

- A **char** in C is one byte (8-bit) in size (16-bit in Java)
 - Will elaborate why 8 bits, 16 bits
- A constant char is specified with single quotes:
 - Regular characters: 'A', 'C', 'z', '0', '#', '\$', ...
 - `char x = 'A';`
 - Special characters: invisible or control chars
 - New line, tab, del
 - Use escape sequence to represent →

59

Same in Java



59

Special Characters

Escape sequence	Meaning
<code>\n</code>	New line
<code>\t</code>	Tab
<code>\0</code>	The null character
<code>\\</code>	The <code>\</code> character
<code>\"</code>	Double quote
<code>\'</code>	Single quote

```
char c = '\t';
char c2 = '\n';
```

Same in Java



60

01100101 01101100 01101100 01101111 00000000				
Internal Representation of Characters				
Dec	Hx	Oct	Char	
0	0	000	NUL (null)	
1	1	001	SOH (start of heading)	
2	2	002	STX (start of text)	
3	3	003	ETX (end of text)	
4	4	004	EOT (end of transmission)	
5	5	005	ENQ (enquiry)	
6	6	006	ACK (acknowledge)	
7	7	007	BEL (bell)	
8	8	010	BS (backspace)	
9	9	011	TAB (horizontal tab)	
10	A	012	LF (NL line feed, new line)	
11	B	013	VT (vertical tab)	
12	C	014	FF (NP form feed, new page)	
13	D	015	CR (carriage return)	
14	E	016	SO (shift out)	
15	F	017	SI (shift in)	
16	10	020	DLE (data link escape)	
17	11	021	DC1 (device control 1)	
18	12	022	DC2 (device control 2)	
19	13	023	DC3 (device control 3)	
20	14	024	DC4 (device control 4)	
21	15	025	NAK (negative acknowledge)	
22	16	026	SYN (synchronous idle)	
23	17	027	ETB (end of trans. block)	
24	18	030	CAN (cancel)	
25	19	031	EM (end of medium)	
26	1A	032	SUB (substitute)	
27	1B	033	ESC (escape)	
28	1C	034	FS (file separator)	
29	1D	035	GS (group separator)	
30	1E	036	RS (record separator)	
31	1F	037	US (unit separator)	

63

01100101 01101100 01101100 01101111 00000000				
Internal Representation of Characters				
<ul style="list-style-type: none"> 0~32 are invisible chars '0' - '9' are encoded consecutively (48~57) 'A' - 'Z' are encoded consecutively (65~90) 'a' - 'z' are encoded consecutively (97~122) Upper letters before lower. Index/encoding difference of 'a' and 'A' is 32, so does 'b' and 'B', 'c' and 'C', ... 8 bits is enough Java uses a bigger character set table 16 bits <u>Unicode</u>, 0~127 are same 				
Dec	Hx	Oct	Html	Chr
32	20	040	#32;	Space
33	21	041	#33;	!
34	22	042	#34;	"
35	23	043	#35;	#
36	24	044	#36;	\$
37	25	045	#37;	%
38	26	046	#38;	&
39	27	047	#39;	'
40	28	050	#40;	(
41	29	051	#41;)
42	2A	052	#42;	*
43	2B	053	#43;	+
44	2C	054	#44;	,
45	2D	055	#45;	-
46	2E	056	#46;	.
47	2F	057	#47;	/
48	30	060	#48;	0
49	31	061	#49;	1
50	32	062	#50;	2
51	33	063	#51;	3
52	34	064	#52;	4
53	35	065	#53;	5
54	36	066	#54;	6
55	37	067	#55;	7
56	38	070	#56;	8
57	39	071	#57;	9
58	3A	072	#58;	:
59	3B	073	#59;	;
60	3C	074	#60;	<
61	3D	075	#61;	=
62	3E	076	#62;	>
63	3F	077	#63;	?

64




Diagram illustrating memory storage and character encoding. It shows two RAM modules and a small icon of two people standing. Below the icon, the characters 'H' and 'e' are shown with their respective ASCII values (72 and 101) and binary representations (01001000 and 01100101). Below this, the characters 'H', 'e', 'l', 'l', 'o' are shown with their ASCII values (72, 101, 108, 108, 111) and their binary representations (01001000, 01100101, 01101100, 01101100, 01101111).

65	41	101	65	A	97	61	141	97	a
66	42	102	66	B	98	62	142	98	b
67	43	103	67	C	99	63	143	99	c
68	44	104	68	D	100	64	144	100	d
69	45	105	69	E	101	65	145	101	e
70	46	106	70	F	102	66	146	102	f
71	47	107	71	G	103	67	147	103	g
72	48	110	72	H	104	68	150	104	h
73	49	111	73	I	105	69	151	105	i
74	4A	112	74	J	106	6A	152	106	j
75	4B	113	75	K	107	6B	153	107	k
76	4C	114	76	L	108	6C	154	108	l
77	4D	115	77	M	109	6D	155	109	m
78	4E	116	78	N	110	6E	156	110	n
79	4F	117	79	O	111	6F	157	111	o
80	50	120	80	P	112	70	160	112	p
81	51	121	81	Q	113	71	161	113	q
82	52	122	82	R	114	72	162	114	r
83	53	123	83	S	115	73	163	115	s
84	54	124	84	T	116	74	164	116	t
85	55	125	85	U	117	75	165	117	u
86	56	126	86	V	118	76	166	118	v
87	57	127	87	W	119	77	167	119	w
88	58	130	88	X	120	78	170	120	x
89	59	131	89	Y	121	79	171	121	y
90	5A	132	90	Z	122	7A	172	122	z

YORK UNIVERSITY

65

Array, later today

65

Characters

- chars are treated in C (and Java) as small integers, char variables and constants are identical to int in arithmetic expressions:
 - char c is converted to its encoding (index in the character set table)

```
char aChar = 'E'; // encoding 69
'E' + 8 // expression with value 69+8 = 77 conv next week
'E' + 'B' // expression with value 69+66 = 135
'E' - 'B' // expression with value 69-66 = 3
```

- Same for other expressions. In relational/logical expression, characters can be compared directly, comparing indexes/encodings

```
aChar == EOF // index == -1? → expr with value 0 (false)
```

```
aChar == 'H' // index == 72? → expr with value 0 (false)
```

```
aChar == '\n' // index == 10? → expr with value 0 (false)
```

```
66 aChar < 'H' // 69 < 72? Earlier in table? → expr with 1 (true)
aChar < 72
```

66

Characters

- `chars` are treated in C (and Java) as small integers, `char` variables and constants are identical to `int` in arithmetic expressions:

- `char c` is converted to its encoding (index in the character set table)

65	41	101	6#65:	A
66	42	102	6#66:	B
67	43	103	6#67:	C
68	44	104	6#68:	D
69	45	105	6#69:	E
70	46	106	6#70:	F
71	47	107	6#71:	G
72	48	110	6#72:	H
73	49	111	6#73:	I
74	4A	112	6#74:	J
75	4B	113	6#75:	K
76	4C	114	6#76:	L

```
int aChar = getchar(); // read 'E' encoding 69
aChar + 8 // expression with value 69+8 = 77
aChar + 'B' // expression with value 69+66 = 135
aChar - 'B' // expression with value 69-66 = 3
```

- Same for other expressions. In relational/logical expression, characters can be compared directly, **comparing indexes/encodings**

```
aChar == EOF // index == -1? → expr with value 0 (false)
```

```
aChar == 'H' // index == 72? → expr with value 0 (false)
```

```
aChar == '\n' // index == 10? → expr with value 0 (false)
```

```
67 aChar < 'H' // 69 < 72? Earlier in table? → expr with 1 (true)
aChar < 72
```

67

char is (represented as) small integers (≤ 256)

```
class CharTest
{
    public static void main(String[] args)
    {
        char aCh = '3'; // encoding 51
        System.out.println(aCh) // 3

        System.out.println(aCh - 0); // 51
        System.out.println(aCh + 4); // 55

        System.out.println(aCh - '0'); // 51-48=3 !
        System.out.println(aCh - '0'+4); // 7

        System.out.println(aCh > 40); // true
        System.out.println(aCh > 'A'); // false
    }
}
```

47	2F	057	6#47:	/
48	30	060	6#48:	0
49	31	061	6#49:	1
50	32	062	6#50:	2
51	33	063	6#51:	3
52	34	064	6#52:	4
53	35	065	6#53:	5
54	36	066	6#54:	6
55	37	067	6#55:	7
56	38	070	6#56:	8
57	39	071	6#57:	9
58	3A	072	6#58:	:
59	3B	073	6#59:	;



68

68

Characters

- Since `chars` are just small integers, `char` variables and constants are identical to `int` in arithmetic expressions:

- `char c` is converted to its encoding (index in the character set table)

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

```
char aCh = '6'; // same as char aCh = 54;
printf("value is %c\n", aCh ); // char 6
printf("value is %d\n", aCh ); // numerical 54
// print encoding
```

```
printf("value is %d\n", aCh + 2 ); // numerical 56
printf("value is %c\n", aCh + 2 ); // char 8
printf("value is %d\n", aCh - 0 ); // 54-0=54
printf("value is %d\n", aCh - '0' ); // 54-48=6
```

69

same in Java

45	2D	055	0#45:
46	2E	056	0#46:
47	2F	057	0#47:
48	30	060	0#48:
49	31	061	0#49:
50	32	062	0#50:
51	33	063	0#51:
52	34	064	0#52:
53	35	065	0#53:
54	36	066	0#54:
55	37	067	0#55:
56	38	070	0#56:
57	39	071	0#57:
58	3A	072	0#58:
59	3B	073	0#59:
60	3C	074	0#60:
61	3D	075	0#61:
62	3E	076	0#62:
63	3F	077	0#63:

69

Characters

- Since `chars` are just small integers, `char` variables and constants are identical to `int` in arithmetic expressions. Some programming idioms that take advantage of this:

```
if(c >= '0' && c <= '9') /*index 48~57, is a digit */
                          (located from '0' to '9')

if(c >= 'a' && c <= 'z') /* low case letter */

if(c >= 'A' && c <= 'Z') /* upper case letter */

if( (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))

if(c >= '0' && c <= '9'){ // c<= 48 c>=57 isdigit(c)
    printf("c is a digit\n");
    printf("numerical value is %d\n", ?);
}
```

70

same in Java

45	2D	055	0#45:
46	2E	056	0#46:
47	2F	057	0#47:
48	30	060	0#48:
49	31	061	0#49:
50	32	062	0#50:
51	33	063	0#51:
52	34	064	0#52:
53	35	065	0#53:
54	36	066	0#54:
55	37	067	0#55:
56	38	070	0#56:
57	39	071	0#57:
58	3A	072	0#58:
59	3B	073	0#59:
60	3C	074	0#60:

70

Characters

- Since `chars` are just small integers, `char` variables and constants are identical to `int` in arithmetic expressions. Some programming idioms that take advantage of this:

```

if(c >= '0' && c <= '9') /*index 48~57, is a digit */
                        (located from '0' to '9')

if(c >= 'a' && c <= 'z') /* low case letter */

if(c >= 'A' && c <= 'Z') /* upper case letter */

if( (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))

if(c >= '0' && c <= '9'){ // c<= 48 c>=57 isdigit(c)
    printf("c is a digit\n");
    printf("numerical value is %d\n", c-'0');
}

```

71

same in Java

c-48 works
but avoid

71

Example

- Upper case letters before lower case letters.
- Encoding difference of 'a' and 'A' is 32, so does 'b' and 'B', 'c' and 'C', 'd' and 'D'...

```

#include<stdio.h>

/*copying input to output with
converting upper-case letters to lower-case */
main(){
    int c; int outC;
    c = getchar();
    while ( c != EOF )
    {
        if (c >= 'A' && c <= 'Z') /* 65~90 upper case letter*/
            outC = c + ('a' - 'A') ; /* = c + 'b' - 'B' */
        else
            outC = c; /* = c + ('c' - 'C') */
        putchar(outC);
        c = getchar(); // read again
    }
}

```

c + 32 works but
not good for portability.
Avoid that!

73


```

main() {
    char le = 'J'; // 74
    while (le <= 'Q') {
        printf ("%d %c %cack %c\n", le, le, le, le+1);
        le++;
    }
}

```

74	J	Jack	K
75	K	Kack	L
76	L	Lack	M
77	M	Mack	N
78	N	Nack	O
79	O	Oack	P
80	P	Pack	Q
7481	Q	Qack	R

same in Java

YORK UNIVERSITY

74

Outline

- Types and sizes
 - Types
 - Constant values (literals)**
 - char** treated as small int
 - int** different bases
 - float
- Array and “strings”
- Expressions
 - Basic operators
 - Type promotion and conversion
 - Other operators
 - Precedence of operators

YORK UNIVERSITY

75

75

Integer Constants

0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
Stored always binary $2^2 \ 2^1 \ 2^0$

- Integer constants can be expressed in three different ways:

1. Decimal [base 10]

- `int x = 31`

same in Java

2. Octal [base 8]

- Start with zero 0

- `int x = 037` (31 in decimal)

same in Java

3. Hexadecimal [base 16]

- Start with 0x or 0X

- `int x = 0x1F` (31 in decimal)

same in Java

*Ways for people to write numbers.
Nothing to do with how the numbers
are stored -- always binary.*



Java also has the 4th way: binary
`int x = 0b00011111`

76

```
cs > home > huiwang > tryC > 21Wteaching > L2 > c binaryLiteral0.c
```

```
3
4  /* salute the world */
5
6  main ()
7
8  int x = 31;
9  int x2 = 037;
10 int x3 = 0x1F;
11
12 printf( "%d\n", x );
13 printf( "%d\n", x2 );
14 printf( "%d\n", x3 );
15
16
17
18
```

same in Java

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ea06 313 % gcc binaryLiteral0.c
ea06 314 % a.out
31
31
31
ea06 315 %
```

77



77

Binary literal is not C standard, although it might work

```
cs > home > huiwang > tryC > 21Wteaching > L2 > C binaryLiteral.c
2  /* import standard IO header files */
3
4  /* salute the world */
5
6  main ()
7  {
8      int x = 30;
9      int x2 = 037;
10     int x3 = 0x1F;
11     int x4 = 0b01111011;
12     printf( "Hello, world\n" );
13 }
14
15
```

okay in Java

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ea57 318 % gcc -pedantic-errors binaryLiteral.c
binaryLiteral.c: In function 'main':
binaryLiteral.c:11:12: error: binary constants are a GCC extension
    int x4 = 0b01111011;
               ^
ea57 319 %
```

78

For your information



78

Octal Notation

- base 8 ... uses 8 symbols

0, 1, 2, 3, 4, 5, 6, 7

- Position represents power of 8
- 1523₈

$$(1 * 8^3) + (5 * 8^2) + (2 * 8^1) + (3 * 8^0) = 851$$

Hexadecimal Notation

- base 16 or 'hex' ... uses 16 symbols

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- Position represents powers of 16

- B65F₁₆ or 0xB65F

$$(11 * 16^3) + (6 * 16^2) + (5 * 16^1) + (15 * 16^0) = 46687$$



79

Others To decimal

• 3 2 8 \rightarrow $3 \cdot 10^2 + 2 \cdot 10^1 + 8 \cdot 10^0 =$ **Decimal 328**
 $10^2 \quad 10^1 \quad 10^0$ $300 \quad + 20 \quad + 8 = 328$

• 1 0 1 \rightarrow $1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$ **Binary 000000000101**
 $2^2 \quad 2^1 \quad 2^0$ $4 \quad + 0 \quad + 1 = 5$

• 3 4 5 \rightarrow $3 \cdot 8^2 + 4 \cdot 8^1 + 5 \cdot 8^0 =$ **Octal 0345**
 $8^2 \quad 8^1 \quad 8^0$ $192 \quad + 32 \quad + 5 = 229$

• 3 4 F \rightarrow $3 \cdot 16^2 + 4 \cdot 16^1 + F \cdot 16^0 =$ **Hex 0x34F**
 $16^2 \quad 16^1 \quad 16^0$ $3 \cdot 256 + 4 \cdot 16 + 15 \cdot 1 =$ **0X34f**
 $768 \quad + 64 \quad + 15 = 847$


80 You should know these conversions.



80

Binary to/from others -- why Hex and Octal

"I want an int with representation 01001100, how to code it in C?"

Java, can do binary `int a = 0b01001100` 

• 0 1 0 0 1 1 0 0 \leftrightarrow $1 \cdot 2^6 + 1 \cdot 2^3 + 1 \cdot 2^2 =$ **Decimal**
 $2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$ $64 \quad + 8 \quad + 4 \quad \text{int a} = 76$

• 0 1 0 0 1 1 0 0 \leftrightarrow **int a = 0114** **Octal**
 $\underbrace{01}_1 \underbrace{00}_1 \underbrace{11}_4 \underbrace{00}_4$ **easier**

• 0 1 0 0 1 1 0 0 \leftrightarrow **int a = 0X4C** **Hex**
 $\underbrace{0100}_4 \underbrace{1100}_{12 \rightarrow C}$ **= 0x4c** **easier**

81 You should know these conversions (both ways).



81

Binary to/from others -- why Hex and Octal

"I want an int with representation 0011011010000110, how to code it in C?"

Java, can do binary `int a = 0b11011010000110`



• 1 1 0 1 1 0 1 0 0 0 0 1 1 0
 $2^{13} 2^{12} 2^{11} 2^{10} 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0$

→ Decimal
`int a = 13958`

$8192+4096+1024+512+128+4+2= 13958$

• 1 1 0 1 1 0 1 0 0 0 0 1 1 0
 3 3 2 0 6

↔ Octal
`int a = 033206`
 easier

• 1 1 0 1 1 0 1 0 0 0 0 1 1 0
 3 6 8 6

↔ Hex
`int a = 0X3686`
`= 0x3686`
 easier

82 You should know these conversions (both ways).



82

Decimal number	Binary representation	Octal representation	Hexadecimal representation
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

`int a=16`
`int a=76`

`int a=0b10000`
`int a=0b1001100`

`int a=020`
`int a=0114`

`int a=0X10`
`int a=0x4C`

Java only

83

Dec	Hx	Oct	...
0	0	000	64 40 100 64: 0
1	1	001	65 41 101 65: A
2	2	002	66 42 102 66: B
3	3	003	67 43 103 67: C
4	4	004	68 44 104 68: D
5	5	005	69 45 105 69: E
6	6	006	70 46 106 70: F
7	7	007	71 47 107 71: G
8	8	010	72 48 110 72: H
9	9	011	73 49 111 73: I
10	A	012	74 4A 112 74: J
11	B	013	75 4B 113 75: K
12	C	014	76 4C 114 76: L
13	D	015	77 4D 115 77: M
14	E	016	78 4E 116 78: N
15	F	017	79 4F 117 79: O
16	10	020	80 50 120 80: P
17	11	021	81 51 121 81: Q
18	12	022	82 52 122 82: R
19	13	023	83 53 123 83: S
20	14	024	84 54 124 84: T
21	15	025	85 55 125 85: U
22	16	026	86 56 126 86: V
23	17	027	87 57 127 87: W
24	18	030	88 58 130 88: X
25	19	031	89 59 131 89: Y
26	1A	032	90 5A 132 90: Z
27	1B	033	91 5B 133 91: [
28	1C	034	92 5C 134 92: \
29	1D	035	93 5D 135 93:]
30	1E	036	94 5E 136 94: ^
31	1F	037	95 5F 137 95: _
			96 60 140 96: `
			97 61 141 97: a
			98 62 142 98: b
			99 63 143 99: c
			100 64 144 100: d
			101 65 145 101: e
			102 66 146 102: f
			103 67 147 103: g
			104 68 150 104: h
			105 69 151 105: i
			106 70 152 106: j
			107 71 153 107: k
			108 72 154 108: l
			109 73 155 109: m
			110 74 156 110: n
			111 75 157 111: o
			112 76 160 112: p
			113 77 161 113: q
			114 78 162 114: r
			115 79 163 115: s
			116 80 164 116: t
			117 81 165 117: u
			118 82 166 118: v
			119 83 167 119: w
			120 84 170 120: x
			121 85 171 121: y
			122 86 172 122: z
			123 87 173 123: {
			124 88 174 124:
			125 89 175 125: }
			126 90 176 126: ~
			127 91 177 127:
			128 92 180 128: 0
			129 93 181 129: 1
			130 94 182 130: 2
			131 95 183 131: 3
			132 96 184 132: 4
			133 97 185 133: 5
			134 98 186 134: 6
			135 99 187 135: 7
			136 100 190 136: 8
			137 101 191 137: 9
			138 102 192 138: A
			139 103 193 139: B
			140 104 194 140: C
			141 105 195 141: D
			142 106 196 142: E
			143 107 197 143: F
			144 108 198 144: G
			145 109 199 145: H
			146 110 200 146: I
			147 111 201 147: J
			148 112 202 148: K
			149 113 203 149: L
			150 114 204 150: M
			151 115 205 151: N
			152 116 206 152: O
			153 117 207 153: P
			154 118 208 154: Q
			155 119 209 155: R
			156 120 210 156: S
			157 121 211 157: T
			158 122 212 158: U
			159 123 213 159: V
			160 124 214 160: W
			161 125 215 161: X
			162 126 216 162: Y
			163 127 217 163: Z
			164 128 220 164: a
			165 129 221 165: b
			166 130 222 166: c
			167 131 223 167: d
			168 132 224 168: e
			169 133 225 169: f
			170 134 226 170: g
			171 135 227 171: h
			172 136 228 172: i
			173 137 229 173: j
			174 138 230 174: k
			175 139 231 175: l
			176 140 232 176: m
			177 141 233 177: n
			178 142 234 178: o
			179 143 235 179: p
			180 144 236 180: q
			181 145 237 181: r
			182 146 238 182: s
			183 147 239 183: t
			184 148 240 184: u
			185 149 241 185: v
			186 150 242 186: w
			187 151 243 187: x
			188 152 244 188: y
			189 153 245 189: z
			190 154 246 190: {
			191 155 247 191:
			192 156 248 192: }
			193 157 249 193: ~
			194 158 250 194:
			195 159 251 195: 0
			196 160 252 196: 1
			197 161 253 197: 2
			198 162 254 198: 3
			199 163 255 199: 4
			200 164 256 200: 5
			201 165 257 201: 6
			202 166 258 202: 7
			203 167 259 203: 8
			204 168 260 204: 9
			205 169 261 205: A
			206 170 262 206: B
			207 171 263 207: C
			208 172 264 208: D
			209 173 265 209: E
			210 174 266 210: F
			211 175 267 211: G
			212 176 268 212: H
			213 177 269 213: I
			214 178 270 214: J
			215 179 271 215: K
			216 180 272 216: L
			217 181 273 217: M
			218 182 274 218: N
			219 183 275 219: O
			220 184 276 220: P
			221 185 277 221: Q
			222 186 278 222: R
			223 187 279 223: S
			224 188 280 224: T
			225 189 281 225: U
			226 190 282 226: V
			227 191 283 227: W
			228 192 284 228: X
			229 193 285 229: Y
			230 194 286 230: Z
			231 195 287 231: [
			232 196 288 232: \
			233 197 289 233:]
			234 198 290 234: ^
			235 199 291 235: _
			236 200 292 236: `
			237 201 293 237: a
			238 202 294 238: b
			239 203 295 239: c
			240 204 296 240: d
			241 205 297 241: e
			242 206 298 242: f
			243 207 299 243: g
			244 208 300 244: h
			245 209 301 245: i
			246 210 302 246: j
			247 211 303 247: k
			248 212 304 248: l
			249 213 305 249: m
			250 214 306 250: n
			251 215 307 251: o
			252 216 308 252: p
			253 217 309 253: q
			254 218 310 254: r
			255 219 311 255: s



83

www.cleavebooks.co.uk/scol/calnumba.htm

← Try it!

Cleave Books

The Number Base Calculator

For detailed instructions on use, and limitations, see below.

Click on [Clear All] to re-start.

[Clear All] [Calculate It]

1 1111	base 2 [0,1]	31	base 10 [0,1,2,3,4,5,6,7,8,9]
1011	base 3 [0,1,2]	29	base 11 [0 to 9, A]
133	base 4 [0,1,2,3]	27	base 12 [0 to 9, A, B]
111	base 5 [0,1,2,3,4]	25	base 13 [0 to 9, A, B, C]
51	base 6 [0,1,2,3,4,5]	23	base 14 [0 to 9, A, B, C, D]
43	base 7 [0,1,2,3,4,5,6]	21	base 15 [0 to 9, A, B, C, D, E]
37	base 8 [0,1,2,3,4,5,6,7]	1F	base 16 [0 to 9, A, B, C, D, E, F]
34	base 9 [0,1,2,3,4,5,6,7,8]	1B	base 20 [0 to 9, A, B, C, D, E, F, G, H, J, K]

Restrictions
Entries limited to equivalent of 10 million.
Only characters indicated on the right may be used.

A = 10 B = 11 C = 12 D = 13 E = 14 F = 15 G = 16 H = 17 J = 18 K = 19

base 2 = binary
base 3 = ternary
base 8 = octal
base 10 = denary or decimal
base 12 = duodecimal
base 16 = hexadecimal

Also a writeup "Number system.pdf" on the course website

YORK UNIVERSITY

84

Integer Constants (finally)

- We can specify type qualifier at the end:

- 'u' or 'U' ⇒ unsigned (int)
- 'l' or 'L' ⇒ long (int)
- nothing ⇒ int

same in Java

- E.g.

5	as an	"(signed) (decimal) int"	5
5U	as an	"unsigned (decimal) int"	5
5L	as a	"(signed) long (int)"	5
5UL or 5uL	as an	"unsigned long (int)"	5
037	as an	"(signed) int (oct)"	decimal: 31
0x32dUL	as an	"unsigned long (int) in hex 32d"	813
059	as an	?	
0x39G2	as an	?	

YORK UNIVERSITY

85

Outline

- Types and sizes
 - Types
 - Constant values (literals)
 - char
 - int
 - **float**
- Array and “strings”
- Expressions
 - Basic operators
 - Type promotion and conversion
 - Other operators
 - Precedence of operators



86

86

Floating Point Constants

- All floating point constants contain a decimal point('.') and/or an exponent ('e' or "E")
 - E.g. 1.532 3e5 4.112e-10
 - 5.3e12 == 5.3×10^{12}
 - `printf("%E %e", 0.00137, 123.025);`
- | | |
|------------|-----------------------------|
| 0.00137 | 1.37 x 10 ⁻³ |
| 15237 | 1.5237 x 10 ⁴ |
| 59000005 | 5.9000005 x 10 ⁷ |
| 123.025 | 1.23025 x 10 ² |
| 0.00005025 | 5.025 x 10 ⁻⁵ |
- 1.370000E-03 1.230250e+02
-
- Floating point constants are of type 'double'
 - Nothing – means “**double**” e.g., `double x = 1.532`
- same in Java
- 'f' or 'F' - means “**float**” e.g. `float x = 1.532f`
- same in Java
- `float x = 1.532` OK
Not OK in Java
Type mismatch: cannot convert from double to float
- 'l' or 'L' - means “**long double**” e.g. `long double x=1.5L`
- same in Java

87

87

Floating literals

```
cs > home > huiwang > tryC > 21Wteaching > L2 > C gravity.c
1  #include <stdio.h>
2
3  int main() {
4      double G = 6.673e-11;    // G is 6.673 x 10^-11,
5      double M = 5.98e24;     // M is the mass of the earth 5.98 x 10^24 (in kg)
6      double accelGravity;
7      double distCenter;
8      printf("Enter distance: ");
9      scanf("%lf", &distCenter);
10
11     accelGravity = (G * M) / (distCenter * distCenter); // (G * M) / (d^2),
12
13     printf("%lf\n", accelGravity);
14     printf("%f\n", accelGravity);
15
16     return 0;
17 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ea57 342 % a.out
Enter distance: 6380000
9.803495
9.803495
ea57 343 % a.out
Enter distance: 6.38e6
9.803495
9.803495
ea57 344 % a.out
Enter distance: 6.38E6
9.803495
9.803495
ea57 345 %
```



88

Summary and plan

- **(Primitive) Types and sizes**
 - **Types:** char, short, int, long, unsigned short, unsigned int, float, double
 - **Constant values (literals)**
 - char
 - int
 - float
- Array and "strings"
- Expressions
 - Basic operators
 - Type promotion and conversion
 - Other operators
 - Precedence of operators

today

Plan for next class

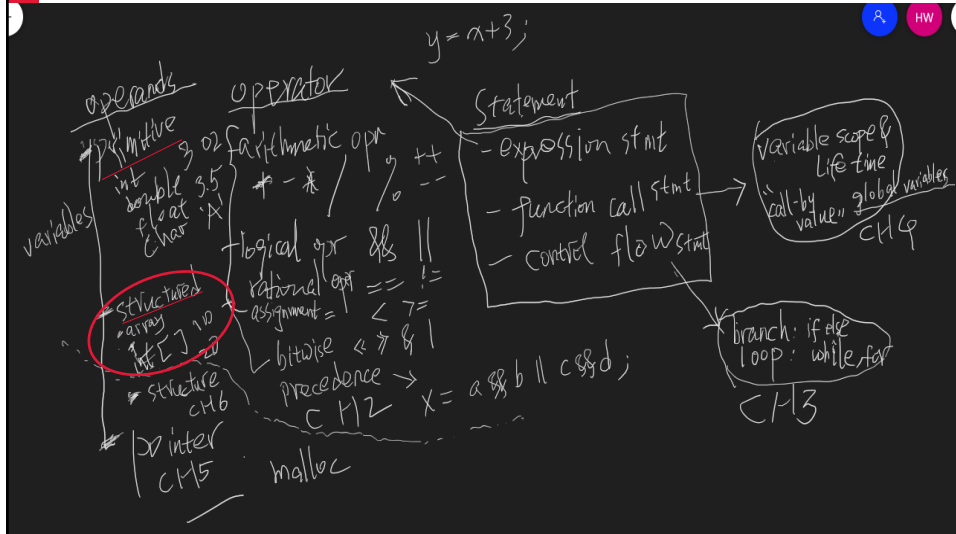
89



89

Roadmap -- How the topics are related

RECALL



91

Arrays

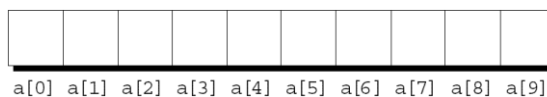
- Indexed list of objects of the **same** type

▪ `int a[10];` -- declare an array of 10 int's
 ▪ `float x[20];` -- declare an array of 20 float's

↑ ↑ ↑
 type name size

- Index numbering starts from 0 (!)

▪ `a[0] ... a[9]`
 ▪ `x[0] ... x[19]` ← array elements



length 10

92

same in Java

92

RECALL

Arrays in Java



```
int[] myIntArray;
```

```
myIntArray = new int[3];
```

or

```
int[] myIntArray= new int[3];
```

Value?
{0, 0, 0};

```
int[] myIntArray = {1, 2, 3};
```

```
int[] myIntArray;
```

```
myIntArray = {1, 2, 3};
```



```
int[] myIntArray = new int[]{1, 2, 3};
```

93



93

Declaring Arrays

```
int[] k = new int[3];
```



```
int[] k = {1, 2, 3};
```

- Declare and initialize

```
-5 122 45623 85 58
```

```
int k[5];          /* each element get some garble value*/
```

```
int k[5] = {1,5,3,2,25}; /* valid 1 5 3 2 25 as Java */
```

```
int k[5]; k = {1,5,3,2,25}; /* invalid, as in Java */
```

```
int k[5] = {1};      /* valid. 1 0 0 0 0 (rest is 0) */
```

```
int k[5] = {1,5};    /* valid. 1 5 0 0 0 (rest is 0) */
```

```
int k[] = {1,5};     /* valid. 1 5 */
```

Interview questions

```
int k[3] ={1,5,3,2,25}  X /* invalid */
```

```
int k[];  X /* invalid "size missing" */
```

94



94

A simple array example

```
#include <stdio.h>

int main () {


    int n[ 10 ]; /* n is an array of 10 integers */
    int i,j;

    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ ) {
        n[ i ] = i + 100; /* set element at location i to i + 100 */
    }

    /* output each array element's value */
    for ( j = 0; j < 10; j++ ) {
        printf("Element[%d] = %d\n", j, n[j] );
    }

    return 0;
}
```

flaw



Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109

95

A simple array example – better version

```
#include <stdio.h>
#define SIZE 10

int main () {

    int n[ SIZE ]; /* n is an array of 10 integers */
    int i,j;

    /* initialize elements of array n to 0 */
    for ( i = 0; i < SIZE; i++ ) {
        n[ i ] = i + 100; /* set element at location i to i + 100 */
    }

    /* output each array element's value */
    for ( j = 0; j < SIZE; ++ ) {
        printf("Element[%d] = %d\n", j, n[j] );
    }

    return 0;
}
```

Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109

96

An example involving array and chars

What does this program do?

```
/*counting digits*/
#include <stdio.h>
#define N 10

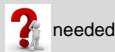
int main () {
    int c, i;
    int digit[N];

    for (i=0; i< N; i++)
        digit[i]=0;

    while ((c = getchar()) != EOF){
        if ( c == '0') digit[0]++;
        elseif ( c == '1') digit[1]++;
        elseif ( c == '2') digit[2]++;
        ...
        elseif ( c == '9') digit[9]++;
    }

    for (i=0; i< N; i++) // has to use loop
        printf ("%d ", digit[i]);

    return 0;
}
```



45	20	055	6#45:	-
46	2E	056	6#46:	.
47	2F	057	6#47:	/
48	30	060	6#48:	0
49	31	061	6#49:	1
50	32	062	6#50:	2
51	33	063	6#51:	3
52	34	064	6#52:	4
53	35	065	6#53:	5
54	36	066	6#54:	6
55	37	067	6#55:	7
56	38	070	6#56:	8
57	39	071	6#57:	9
58	3A	072	6#58:	:
59	3B	073	6#59:	:
60	3C	074	6#60:	<



Simpler
code? Lab2

97

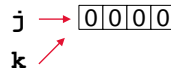
Accessing Arrays

- In C, you can only assign to array members
 - This means you **cannot copy/assign to a whole array**:

```
int i, k[4], j[4];
for (i=0; i<4; i++)
    j[i]= 0;      /* another way? int j[4]={0} */

k = j; ✗ /* invalid */ /* perfectly valid in Java */

for (i=0; i<4; i++)    i=0;
    k[i] = j[i];        while(i<4)
                        or {
                            k[i] = j[i];
                            i++;
                        }
```



```
for (i=0; i<10; i++)
    k[i] = j[i];
```

Compiles, may or may not crash
no boundary checking



99 k=j k==j explain later

99

Summary and plan

- **(Primitive) Types and sizes**
 - **Types:** char, short, int, long, unsigned short, unsigned int, float, double
 - **Constant values (literals)**
 - char
 - int
 - float
- Array and **“strings”**
- Expressions
 - Basic operators
 - Type promotion and conversion
 - Other operators
 - Precedence of operators

today

100

