EECS2031 AC Software Tools

F 2021

Sep 17,20, 2021 Lecture 4

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

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

Same in Java

Special Characters

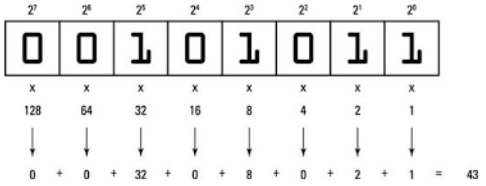
Escape sequence	Meaning
\n	New line
\t	Tab
\0	The null character
\\	The \ character
\"	Double quote
\'	Single quote

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

Same in Java

Internal representation of characters

```
int i =  43;

char a = 'A';
```



How to represent 'A' using 0s and 1s

YORK UNIVERSITÉ UNIVERSITY

61

61

01100101 | 01101100 | 01101100 | 01101111 | 00000000

Internal Representation of characters

- characters as 1/0 bits. So they are stored as (small) integer values, interpreted according to the character set encoding (usually ASCII, 7 bits for 128 characters),
  - `'a'` has encoding `97`, `'0'` has `48`, `'9'` has `57`
- Escape sequences are integers too
  - e.g. `'\n'` has `10` (newline character)
    `'\t'` has `9` (horizontal tab)
- Special escape: `'\0'`
  has encoding 0 - the null character

YORK UNIVERSITÉ UNIVERSITY

62

62

01100101	01101100	01101100	01101111	00000000

Internal Representation of Characters

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

63

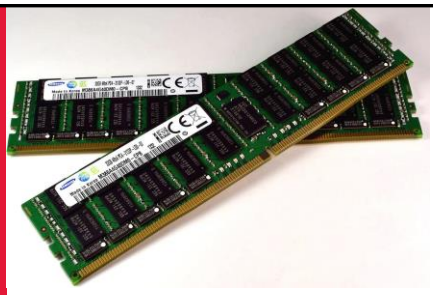
01100101	01101100	01101100	01101111	00000000

Internal Representation of Characters

- 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 Unicode, 0~127 are same

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040		Space	64	40	100	@	@	96	60	140	`	`
33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

64



'H'
72
01001000

'e'
101
01100101

'H'	'e'	'l'	'l'	'o'
72	101	108	108	111
01001000	01100101	01101100	01101100	01101111

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

65 Array, later today

YORK UNIVERSITÉ UNIVERSITY

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)

65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f
71	47	107	G	G	103	67	147	g	g
72	48	110	H	H	104	68	150	h	h
73	49	111	I	I	105	69	151	i	i
74	4A	112	J	J	106	6A	152	j	j
75	4B	113	K	K	107	6B	153	k	k
76	4C	114	L	L	108	6C	154	l	l

```
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

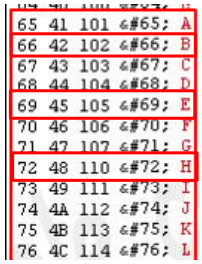
```
aChar == EOF     // index == -1?  ➔ expr with value 0 (false)

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

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

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

66

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)




```
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

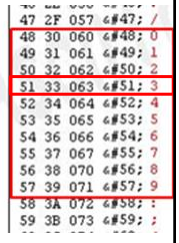
```
aChar == EOF     // index == -1?  ➔ expr with value 0 (false)

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

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

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

67

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
   }
}
```

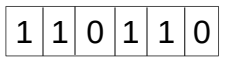


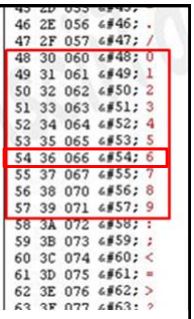
68

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)






```
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
```

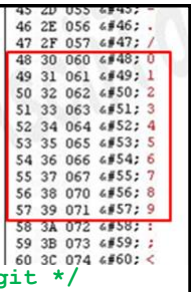
69

same in Java

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", );
}
```

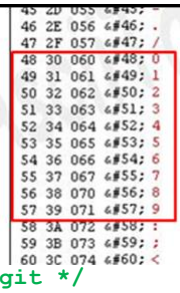
70

same in Java

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');
}
```

c-48 works but avoid

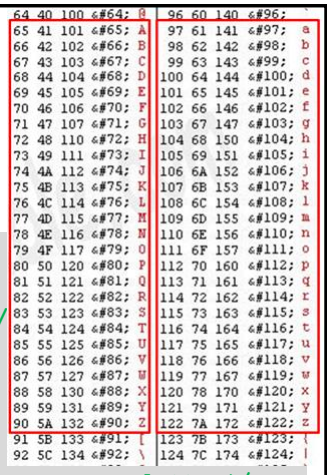
same in Java

71

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                          /* = c + ('c' - 'C') */
            outC = c;                    .....
                                      /* = c + 'z' - 'Z'  */
        putchar(outC);                /*  = tolower(c) */

        c = getchar(); // read again
    }
```

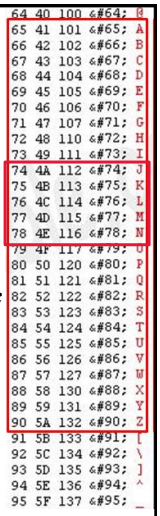
c + 32 works but not good for portability. Avoid that!

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
81  Q   Qack  R
```

same in Java



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

75

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

- Integer constants can be expressed in <u>three</u> 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`

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 %
```

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
- 1523_8

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

  10 11 12 13 14 15
- Position represents powers of 16
- $B65F_{16}$ or 0xB65F

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

79

Others To decimal

- 3 2 8
  10^2 10^1 10^0
  → $3*10^2 + 2*10^1 + 8*10^0 =$ 300 + 20 + 8 = 328
  Decimal 328

- 1 0 1
  2^2 2^1 2^0
  → $1*2^2 + 0*2^1 + 1*2^0 =$ 4 + 0 + 1 = 5
  0000000000101
  Binary

- 3 4 5
  8^2 8^1 8^0
  → $3*8^2 + 4*8^1 + 5*8^0 =$ 192 + 32 + 5 = 229
  Octal 0345

- 3 4 F
  16^2 16^1 16^0
  → $3*16^2 + 4*16^1 + F*16^0 =$
  $3*256 + 4*16 + 15*1 =$
  768 + 64 + 15 = 847
  Hex 0x34F
  0X34f

80 You should know these conversions.

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
  2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
  ↔ $1*2^6 + 1*2^3 + 1*2^2 =$ 64 + 8 + 4
  Decimal
  `int a = 76`

- 01 001 100
  1 1 4
  ↔ **`int a = 0114`**
  Octal
  easier

- 0100 1100
  4 12 → C
  ↔ **`int a = 0X4C`**
  `= 0x4c`
  Hex
  easier

81 You should know these conversions (both ways).

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

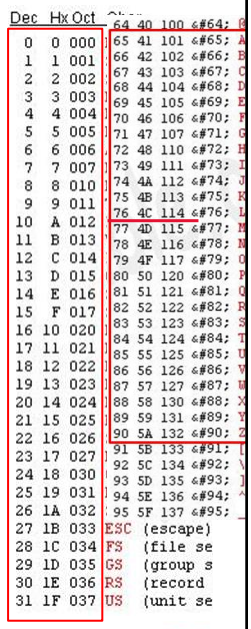
⟺ Octal `int a = 033206` easier

- 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).

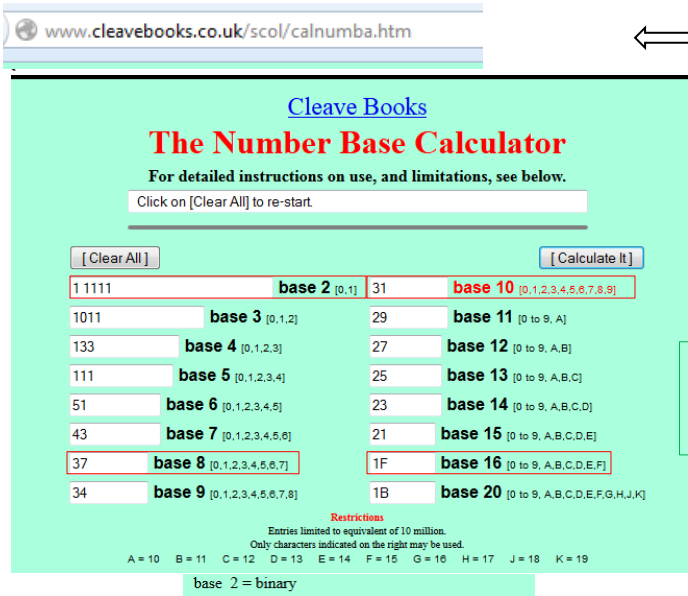
Decimal number	Binary representation	Octal representation	Hexadecimal representation
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10



Dec	Hx	Oct	
0	0	000	
1	1	001	
2	2	002	
3	3	003	
4	4	004	
5	5	005	
6	6	006	
7	7	007	
8	8	010	
9	9	011	
10	A	012	
11	B	013	
12	C	014	
13	D	015	
14	E	016	
15	F	017	
16	10	020	
17	11	021	
18	12	022	
19	13	023	
20	14	024	
21	15	025	
22	16	026	
23	17	027	
24	18	030	
25	19	031	
26	1A	032	
27	1B	033	ESC (escape)
28	1C	034	FS (file se
29	1D	035	GS (group s
30	1E	036	RS (record
31	1F	037	US (unit se

int a=16	int a=0b10000	int a=020	int a=0X10
int a=76	int a=0b1001100	int a=0114	int a=0x4C

Java only



⇦ Try it!

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

84

YORK UNIVERSITÉ 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	?

85

YORK UNIVERSITÉ 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);`

1.370000E-03 1.230250e+02

0.00137	1.37×10^{-3}
15237	1.5237×10^{4}
59000005	5.9000005×10^{7}
123.025	1.23025×10^{2}
0.00005025	5.025×10^{-5}

- 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 graviy.c

```c
#include <stdio.h>

int main() {
    double G = 6.673e-11;   // G is 6.673 x 10^-11,
    double M = 5.98e24;     // M is the mass of the earth 5.98 x 10^24 (in kg)
    double accelGravity;
    double distCenter;
    printf("Enter dirstance: ");
    scanf("%lf", &distCenter);

    accelGravity = (G * M) / (distCenter * distCenter);  // (G * M) / (d^2),

    printf("%lf\n", accelGravity);
    printf("%f\n", accelGravity);

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ea57 342 % a.out
Enter dirstance: 6380000
9.803495
9.803495
ea57 343 % a.out
Enter dirstance: 6.38e6
9.803495
9.803495
ea57 344 % a.out
Enter dirstance: 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**

today

- Array and “strings”

- Expressions
  - Basic operators
  - Type promotion and conversion
  - Other operators
  - Precedence of operators

Plan for next class

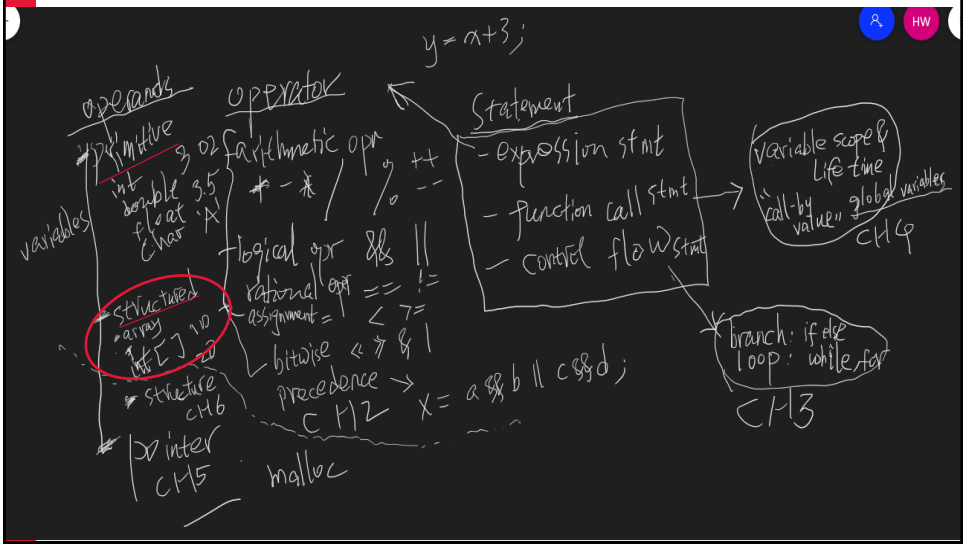
89

YORK UNIVERSITÉ UNIVERSITY

89

Roadmap -- How the topics are related

RECALL



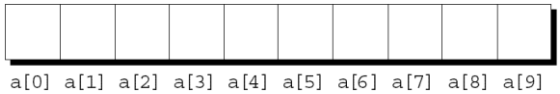
91

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

same in Java

92

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

YORK UNIVERSITÉ UNIVERSITY

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

94

YORK UNIVERSITÉ UNIVERSITY

94

A simple array example

```c
#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;
}
```

```
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
```

flaw

95

A simple array example – better version

```c
#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
```

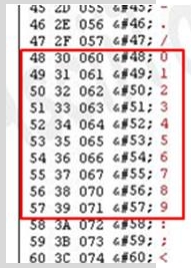
96

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;
}
```

needed



Simpler code? Lab2

97

97

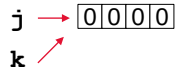
Accessing Arrays

- 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++)
   k[i] = j[i];
```

or

```
i=0;
while(i<4)
{
   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**

today

- Array and "**strings**"

- Expressions
  - Basic operators
  - Type promotion and conversion
  - Other operators
  - Precedence of operators