

# Types, Operators and Expressions

CSE 2031  
Fall 2010

9/26/2010 12:59 PM

1

## Variable Names (2.1)

- Combinations of letters, numbers, and underscore character ( `_` ) that
  - do not start with a number;
  - are not a keyword.
- Upper and lower case letters are distinct ( $x \neq X$ ).
- Examples: Identify valid and invalid variable names  
`abc`, `aBc`, `abc5`, `aA3_`, `char`, `_360degrees`,  
`5sda`, `my_index`, `_temp`, `string`, `struct`,  
`pointer`

2

## Variable Names: Recommendations

- Don't begin variable names with underscore \_
- Limit the length of a variable name to 31 characters or less.
- Function names, external variables: may be less than 31 characters allowed, depending on systems.
- Lower case for variable names.
- Upper case for symbolic constants
  - `#define MAX_SIZE 100`
- Use short names for local variables and long names for external variables.

3

## Data Types and Sizes (2.2)

4 basic types in C

- **char** – characters (8 bits)
- **int** – integers (either 16 or 32 bits)
- **float** – single precision floating point numbers (4 bytes)
- **double** – double precision floating point numbers (8 bytes)

4

## Qualifiers

- **signed** char **sc**; /\* -127 - +128 \*/
- **unsigned** char **uc**; /\* 0 - +255 \*/
- **short** **s**; /\* 16 bits, -32,768 - +32,767 \*/
  - **short** int **s**;
- **long** **counter**; /\* 32 bits \*/
  - **long** int **counter**;
  - **int** is either 16 or 32 bits, depending on systems.
- **signed** int **sint**; /\* same as **int sint**; \*/
- **unsigned** int **uint**;
  - 0 - +4,294,967,295, assuming 4-byte int
- **long** double **ld**; /\* 12 bytes \*/

5

## Qualifiers (cont.)

- <limits.h> and <float.h> contain
  - symbolic constants for all of the above sizes,
  - other properties of the machine and compiler.
- To get the size of a type, use **sizeof( )**  
**int\_size = sizeof( int );**

6

# Characters

- 8 bits
- Included between 2 single quotes  
`char x = 'A'`
- Character string: enclosed between 2 double quotes  
`"This is a string"`
- Note: 'A' ≠ "A"



- `c = '\012'` /\* 10 decimal; new line character \*/

7

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

Source: [www.LookupTables.com](http://www.LookupTables.com)

## Constants (2.3)

- Numeric constants
- Character constants
- String constants
- Constant expressions
- Enumeration constants

9

## Integer Constants

- Decimal numbers
  - 123487
- Octal: starts with 0 (zero)
  - 0654
- Hexadecimal: starts with 0x or 0X
  - 0x4Ab2, 0X1234
- long int: suffixed by L or l
  - 7L, 106l
- unsigned int: suffixed by U or u
  - 8U, 127u

10

## Floating-point Constants

15.75

1.575E1 /\* = 15.75 \*/

1575e-2 /\* = 15.75 \*/

-2.5e-3 /\* = -0.0025 \*/

25E-4 /\* = 0.0025 \*/

100.0L /\* long double \*/

100.0F /\* float \*/

- If there is no suffix, the type is considered **double** (8 bytes).
- To specify **float** (4 bytes), use suffix F or f.
- To specify **long double** (12 bytes), use suffix L or l.

- You can omit the integer portion of the floating-point constant.

.0075e2

0.075e1

.075e1

75e-2

11

## Numeric Constants

• 2010

• 100000

• 729**L** or 729**l**

• 2010**U** or 2010**u**

• 20628**UL** or 20628**ul**

• 24.7 or 1**e**-2

• 24.7**F** or 24.7**f**

• 24.7**L** or 24.7**l**

• 037

• 0**x**1**f**, 0**X**1**f**, 0**x**1**F**

• 0**X**F**U**L

• int

• will be taken as long

• long (int)

• unsigned

• unsigned long

• double

• float

• long double

• octal (= 31 decimal)

• hexadecimal (= 31)

• What is this?

12

## Character Constants

`'x'`

- letter `x`

`'2'`

- numeric value 50

`'\0'`

- NULL char, value 0

`#define NEW_LINE '012'`

- octal, 10 in decimal

`#define NEW_LINE '12'`

- `'ooo'` 1 to 3 octal digits

`#define SPACE '\x20'`

- hex, 32 in decimal

13

## Escape Sequences

<code>\a</code>	alert (bell) character	<code>\\</code>	backslash
<code>\b</code>	backspace	<code>\?</code>	question mark
<code>\f</code>	formfeed	<code>\'</code>	single quote
<code>\n</code>	newline	<code>\"</code>	double quote
<code>\r</code>	carriage return	<code>\ooo</code>	octal number
<code>\t</code>	horizontal tab	<code>\xhh</code>	hexadecimal number
<code>\v</code>	vertical tab		

14

## String Constants

```
"hello, world\n"
```

```
"" /* empty string */
```

```
\\" /* double quote character */
```

```
"hello," " world" same as "hello, world"
```

- concatenated at compile time
- useful for splitting up long strings across several source lines.

15

## Constant Expressions

- Expressions that involve only constants.
- Evaluated during compilation.

```
#define MAXLINE 1000  
char line[MAXLINE+1];
```

```
#define LEAP 1 /* in leap years */  
int days[31+28+LEAP+31+30+31+30+31+31+30+31+30+31];
```

16



## Enumeration Constants

```
enum boolean { NO, YES };
```

- The first name in an enum has value 0, the next 1, and so on, unless explicit values are specified.

```
enum colours { black, white, red, blue, green };
```

```
enum escapes { BELL = '\a', BACKSPACE = '\b', TAB =  
  '\t', NEWLINE = '\n', VTAB = '\v', RETURN = '\r'  
};
```

- If not all values are specified, unspecified values continue the progression from the last specified value.

```
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,  
  AUG, SEP, OCT, NOV, DEC };  
/* FEB = 2, MAR = 3, etc. */
```

17

## Limits

- File `limits.h` provides several constants
  - `char` `CHAR_BIT`, `CHAR_MIN`, `CHAR_MAX`,  
`SCHAR_MIN`, ...
  - `int` `INT_MIN`, `INT_MAX`, `UINT_MAX`
  - `long` `LONG_MIN`, ...
- You can find `FLOAT_MIN`, `DOUBLE_MIN`, ... in  
`<float.h>`

18

## Declarations (2.4)

- All variables must be declared before use (certain declarations can be made implicitly by content).
- A variable may also be initialized in its declaration.

```
char esc = '\\';  
int i = 0;  
int limit = MAXLINE+1;  
float eps = 1.0e-5;
```

19

## Qualifier **const**

- Indicates that the value of a variable will not be changed.
- For an array: the elements will not be altered.

```
const double e = 2.71828182845905;  
const char msg[] = "warning: ";
```

- Used with array arguments, to indicate that the function does not change that array.

```
int strlen( const char[] );
```

- Note: The result is implementation-defined if an attempt is made to change a **const**.

20

## Arithmetic Operators (2.5)

`+ - * / %`

Examples:

```
abc = x + y * z;
```

```
j = a % i;
```

```
++x;
```

```
x++;
```

```
x += 5; /* x = x + 5; */
```

```
y /= z; /* y = y / z */
```

What is `x *= y + 1` ?

21

## Precedence and Associativity

Operators	Associativity
<code>() [] -&gt; .</code>	left to right
<code>! ~ ++ -- + - * (type) sizeof</code>	right to left
<code>* / %</code>	left to right
<code>+ -</code>	left to right
<code>&lt;&lt; &gt;&gt;</code>	left to right
<code>&lt; &lt;= &gt; &gt;=</code>	left to right
<code>== !=</code>	left to right
<code>&amp;</code>	left to right
<code>^</code>	left to right
<code> </code>	left to right
<code>&amp;&amp;</code>	left to right
<code>  </code>	left to right
<code>? :</code>	right to left
<code>= += -= *= /= %= &amp;= ^=  = &lt;&lt;= &gt;&gt;=</code>	right to left
<code>,</code>	left to right

22

## Type Conversion (2.7)

- `float f; int i;` What is the type of `f+i` ?
- General rule: convert a “narrower” operand into a “wider” one without losing information.
- So `i` is converted to float before the addition.
- `char` may be freely used in arithmetic expressions.

```
/* lower: convert c to lower case; ASCII only */
int lower(int c)
{
    if (c >= 'A' && c <= 'Z')
        return c - 'A' + 'a';
    else return c;
}
```

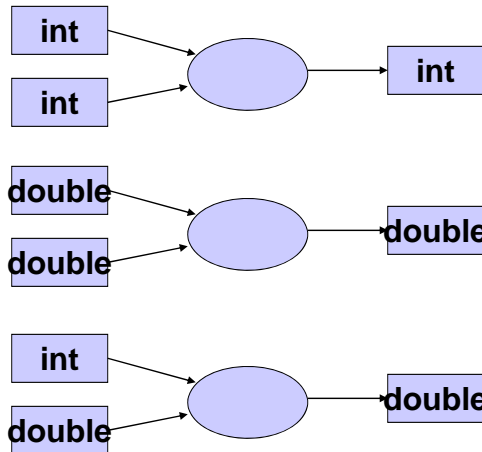
23

## Arithmetic Conversion

- When a binary operator has operands of different types, the “lower” type is *promoted* to the “higher” type before the operation proceeds.
- If either operand is long double, convert the other to long double.
- Otherwise, if either operand is double, convert the other to double.
- Otherwise, if either operand is float, convert the other to float.
- Otherwise, convert char and short to int.
- Then, if either operand is long, convert the other to long.

24

## Arithmetic Conversion: Examples



```
int a=5, b=2, c;  
double x, y = 2;
```

```
x = a/b;  
    // x = 2.0  
c = a/b;  
    // c = 2  
x = a/y;  
    // x = 2.5  
c = a/y;  
    // c = 2
```

25

## More Examples

- 17 / 5
  - 3
- 17.0 / 5
  - 3.4
- 9 / 2 / 3.0 / 4
  - 9 / 2 = 4
  - 4 / 3.0 = 1.333
  - 1.333 / 4 = 0.333

26

## Type Conversion: More Rules

- Conversions take place **across assignments**; the value of the right side is converted to the type of the left, which is the type of the result.

- Example:

```
int a;  
float x = 7, y = 2;  
a = x / y;
```

- `float` to `int` causes truncation of any fractional part.

- Example:

```
float x, y = 2.7;  
int i = 5;  
x = i; /* x = 5.0 */  
i = y; /* i = 2 */
```

27

## Type Conversion: Even More Rules

- Longer integers are converted to shorter ones or to chars by dropping the excess high-order bits.

```
int i;  
char c;  
i = c;  
c = i;  
/* c unchanged */
```

```
int i;  
char c;  
c = i;  
i = c;  
/* i may be changed */
```

28

## Casting

```
int A = 9, B = 2;
double x;
x = A / B; /* x is 4.0 */
x = A / (double)B; /* C is 4.5 */
```

```
int n;
sqrt(double(n))
```

Doesn't change the value of B,  
just changes the type to double

- The cast operator has the same high precedence as other unary operators.

29

## Increment and Decrement Operators (2.8)

- ++ or --
- Placing in front: incrementing or decrementing occurs **BEFORE** value assigned

i = 2 and k = 1

k = ++i;

i = i + 1; 3
k = i; 3

k = --i;

i = i - 1; 1
k = i; 1

- Placing after: occurs **AFTER** value assigned

i = 2 and k = 1

k = i++;

k = i; 2
i = i + 1; 3

k = i--;

k = i; 2
i = i - 1; 1

30

# Precedence and Associativity

Operators	Associativity
() [] -> .	left to right
! ~ ++ -- + - *(type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
	left to right
&&	left to right
	left to right
?:	right to left
= += -= *= /= %= &= ^=  = <<= >>=	right to left
,	left to right

31

## Examples

```
int a=2, b=3; c=5, d=7, e=11, f=3;
```

```
f += a/b/c;           3
```

```
d -= 7+c*--d/e;      -3
```

```
d = 2*a%b+c+1;       7
```

```
a += b +=c += 1+2;   13
```

32



## Relational and Logic Operators (2.6)

- Relational operators:

> >= < <=  
== !=

- Logical operators:

! && ||

- Evaluation stops as soon as the truth or falsehood of the result is known.

```
for ( i=0;
      i < lim-1 &&
      (c=getchar()) != '\n' &&
      c != EOF;
      ++i )
    s[i] = c;

if (valid == 0)
/* same as */
if (!valid)
```

33

## Boolean Expressions

- False is 0; any thing else is 1 (true).
- Write

```
if (!valid)
```

instead of

```
if (valid == 0)
```

34

## Bitwise Operators (2.9)

- Work on individual bits  
`&`   `|`   `^`   `~`
  - Examples:  
`short int i=5, j=8;`  
`k=i&j;`  
`k=i|j;`  
`k=~j;`
  - Application: bit masking
- ```
a = 1;  
b = 2;  
c = a & b; /*c = 0*/  
d = a && b; /*d = 1*/  
  
n = n & 0177;  
x = x | SET_ON;
```

35

## Bit Shifting

- `x<<y` means shift x to the left y times.
    - equivalent to multiplication by  $2^y$
  - `x>>y` means shift x to the right y bits.
    - equivalent to division by  $2^y$
  - Left shifting 3 many times:
- |    |       |
|----|-------|
| 0  | 3     |
| 1  | 6     |
| 2  | 12    |
| 3  | 24    |
| 4  | 48    |
| 5  | ...   |
| 13 | 49512 |
| 14 | 32768 |

36

## Right Shifting

- It could be logical (0) or arithmetic (signed)
- If unsigned, 0; if signed undefined in C

```
unsigned int i = 714;
357 178 89 44 22 11 5 2 1 0
```

- What if  $i = -714$  ?

```
-357 -178 -89 ... -3 -2 -1 -1 -1 -1
```

37

## Bitwise Operators: Examples

```
x = x & ~077;
```

sets the last six bits of x to zero.

```
/* getbits: get n bits from position p */  
unsigned getbits(unsigned x, int p, int n)  
{  
    return (x >> (p+1-n)) & ~(~0 << n);  
}
```

38

## Assignment Operators / Expressions (2.10)

- `A *= B;` // equivalent to
- `A = (A) * (B);` // note the parentheses
- Can be used with: `+ - * / % << >> & ^ |`

```
yyval[yyvsp[p3+p4] + yyv[p1]] += 2
```

```
/* bitcount: count 1 bits in x */
int bitcount(unsigned x) {
    int b;
    for ( b = 0; x != 0; x >>= 1 )
        if ( x & 01 )
            b++;
    return b;
}
```

39

## Conditional Expressions (2.11)

`exp1 ? exp2 : exp3`

- If `exp1` is true, the value of the conditional expression is `exp2`; otherwise, `exp3`.

```
z = (a > b) ? a : b; /* z = max (a, b) */
```

- If `expr2` and `expr3` are of different types, the type of the result is determined by the conversion rules discussed earlier.

```
int n; float f;
(n > 0) ? f : n
/* result of type float in either case */
```

40

## Conditional Expressions: Advantage

- Succinct code

- Example 1:

```
for (i = 0; i < n; i++)  
    printf("%6d%c", a[i],  
          (i%10==9 || i==n-1) ? '\n' : ' ');
```

- Example 2:

```
printf("You have %d item%s.\n", n,  
      n==1 ? "" : "s");
```

41

## Precedence and Order of Evaluation (2.12)

| Operators                         | Associativity |
|-----------------------------------|---------------|
| () [] -> .                        | left to right |
| ! ~ ++ -- + - * (type) sizeof     | right to left |
| * / %                             | left to right |
| + -                               | left to right |
| << >>                             | left to right |
| < <= > >=                         | left to right |
| == !=                             | left to right |
| &                                 | left to right |
| ^                                 | left to right |
|                                   | left to right |
| &&                                | left to right |
|                                   | left to right |
| ?:                                | right to left |
| = += -= *= /= %= &= ^=  = <<= >>= | right to left |
| ,                                 | left to right |

42



## Next time ...

- Control Flow (Chapter 3, C book)
- Functions and program structures (Chapter 4, C book)
- Basic UNIX (Chapter 1, UNIX book)