ESc101: Variable Types

Instructor: Krithika Venkataramani
Semester 2, 2011-2012

The content of these slides are taken from the lecture slides of Prof. Arnab Bhattacharya, Prof. R.K. Ghosh, Prof. Dheeraj Sanghi and Prof. Manindra Agrawal

Variables

- Variables signify data that may be modified
- Name of a variable can contain letters, digits and underscore _
  - Example: i, y2k, big_name, bigger_name_2
- Case-sensitive: camel, CAMEL and CaMeL are different
- Name cannot start with a digit
  - Example: 1d is not valid
- Name can start with an underscore, but do not do so
  - Example: avoid valid names such as _bad
- Certain keywords are special
  - They are reserved and cannot be used
  - Example: main, if

Types of Variables

- Each variable ‘type’ represents the domain of values
  - Integer: int or char
  - Character: char
  - Boolean: int or char
  - real: double or float
- However, they can store only a subset of the domain
  - int can store numbers from -2^{31} to 2^{31} -1
- Initial values of variables are specified as constants of the same type
  - int i = 0;
  - double d = 1.4;
  - char c = 'A'

Example Program to Add two numbers

```
/* Program to add two numbers */
#include <stdio.h>     // Include headers
void main()     // Main function
{
int a=2, b=3, c;          // Declare variables
scanf("%d", &a);       // Read 'a' from keyboard

scanf("%d", &b);       // Read 'b' from keyboard
c = a + b;
printf("%d\n", c);       // Write 'c' to screen
}
```

Different Data Types

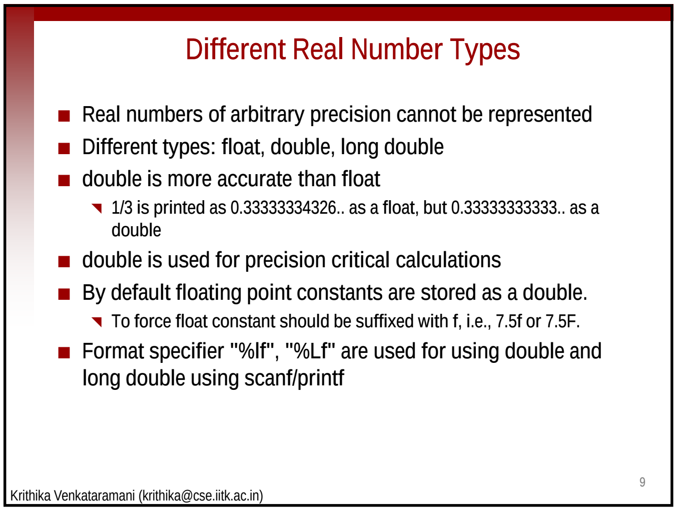
- Different types are needed because one type is not suitable for representing data of another type.
- Mixing types may result in precision loss, overflow, underflow
- Application performance suffers while performing numerically intensive computation if inappropriate data types are used.
- Exceptions must be handled explicitly or they lead to errors.
- Use of appropriate type is important both for efficiency and correctness

Integer Types

- Two different int types: signed and unsigned
- Maximum signed int in 16 bit: 0111111111111111, i,e., 2^{15} - 1
- Maximum unsigned int in 16 bit: 1111111111111111, i.e., 2^{16} - 1
- Possible types to suit our needs are:
  - short int, unsigned short int, unsigned int, long int, unsigned long int.

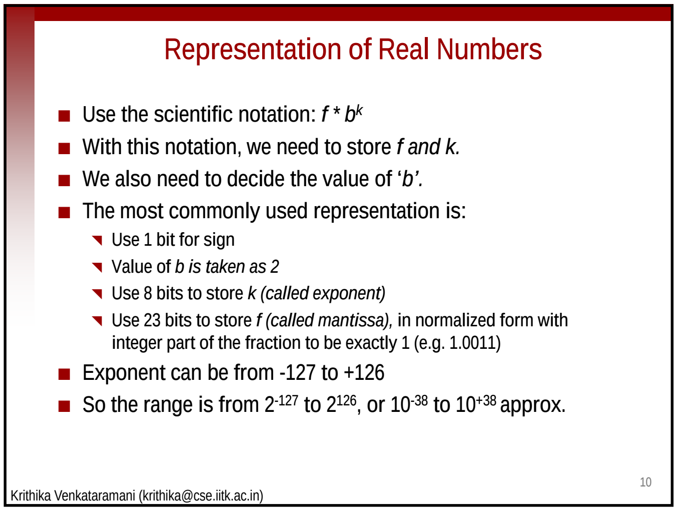
Integer Representation

- We represent integers as binary numbers
  - For example: 56 will be stored as
    - 00111000
- How do we store negative numbers?
  - 1st bit of memory is usually the sign bit
  - In case of 8 bit memory, only 7 bits are for magnitude.
  - Similarly 32-bit memory would have 31 bits for magnitude
  - Hence the largest positive integer that can be stored in an integer variable on our PCs is: $2^{31} – 1$.
  - Smallest number:
- There are variations on storing magnitude
- Overflow: Trying to store numbers outside the range



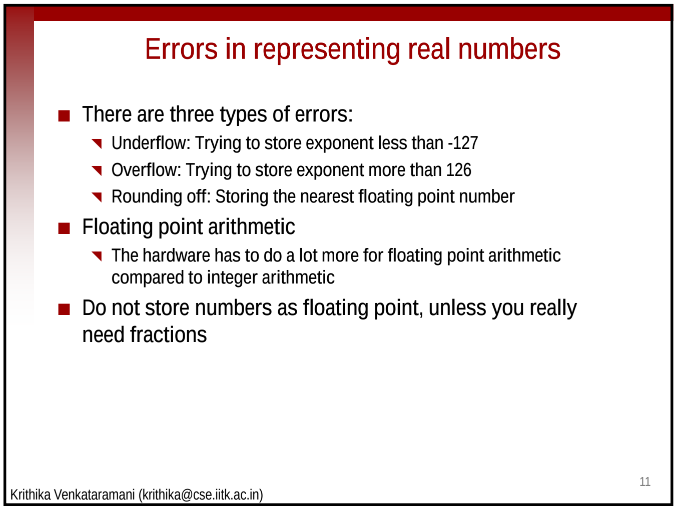
Different Real Number Types

- Real numbers of arbitrary precision cannot be represented
- Different types: float, double, long double
- double is more accurate than float
  - 1/3 is printed as 0.33333334326.. as a float, but 0.33333333333.. as a double
- double is used for precision critical calculations
- By default floating point constants are stored as a double.
  - To force float constant should be suffixed with f, i.e., 7.5f or 7.5F.
- Format specifier "%lf", "%Lf" are used for using double and long double using scanf/printf



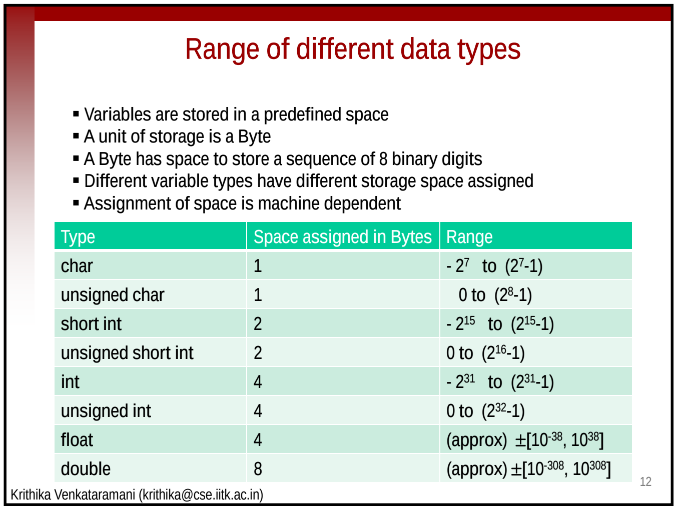
Representation of Real Numbers

- Use the scientific notation: $f * b^k$
- With this notation, we need to store *f and k.*
- We also need to decide the value of '*b'.*
- The most commonly used representation is:
  - Use 1 bit for sign
  - Value of *b is taken as 2*
  - Use 8 bits to store *k (called exponent)*
  - Use 23 bits to store *f (called mantissa),* in normalized form with integer part of the fraction to be exactly 1 (e.g. 1.0011)
- Exponent can be from -127 to +126
- So the range is from 2^{-127} to 2^{126}, or 10^{-38} to 10^{+38} approx.



Errors in representing real numbers

- There are three types of errors:
  - Underflow: Trying to store exponent less than -127
  - Overflow: Trying to store exponent more than 126
  - Rounding off: Storing the nearest floating point number
- Floating point arithmetic
  - The hardware has to do a lot more for floating point arithmetic compared to integer arithmetic
- Do not store numbers as floating point, unless you really need fractions



Range of different data types

- Variables are stored in a predefined space
- A unit of storage is a Byte
- A Byte has space to store a sequence of 8 binary digits
- Different variable types have different storage space assigned
- Assignment of space is machine dependent

Type	Space assigned in Bytes	Range
char	1	$- 2^7$ to (2^7-1)
unsigned char	1	0 to (2^8-1)
short int	2	$- 2^{15}$ to $(2^{15}-1)$
unsigned short int	2	0 to $(2^{16}-1)$
int	4	$- 2^{31}$ to $(2^{31}-1)$
unsigned int	4	0 to $(2^{32}-1)$
float	4	(approx) $\pm[10^{-38}, 10^{38}]$
double	8	(approx) $\pm[10^{-308}, 10^{308}]$

Input and output of variables

- Correct type specification must be used

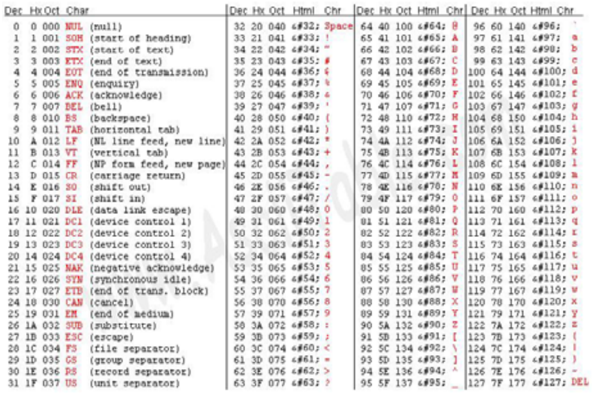
Type	Format Specifier
char	%c
int	%d
unsigned int	%u
float	%f, %g, %e
double	%lf
long double	%Lf

- scanf is for input
  - Format: scanf(“<specification>”, &<name>);
  - E.g. c is a char: scanf(“%c”, &c);
- printf is for output
  - Format: printf(“<specification>”, <name>);
  - E.g. c is a char: printf(``%c", c);

Character type

- Variable type ‘char’ used for representing characters
- Characters are special integers of much shorter size
  - Only 256 characters can be represented
- Digits 0-9 are *not* represented by 00000000 - 00001001
- 0-9 represented by a continuous sequence
- Similarly A-Z (a-z) also represented by a continuous sequence
- ASCII character set is most widely used
  - specifies a standard that maps characters to numbers 0-127
  - Extended ASCII assigns symbols to numbers 128-255
  - ASCII and Extended ASCII use 1 Byte for storage
- Unicode includes characters from all languages of the world
  - Unicode uses 2 Bytes

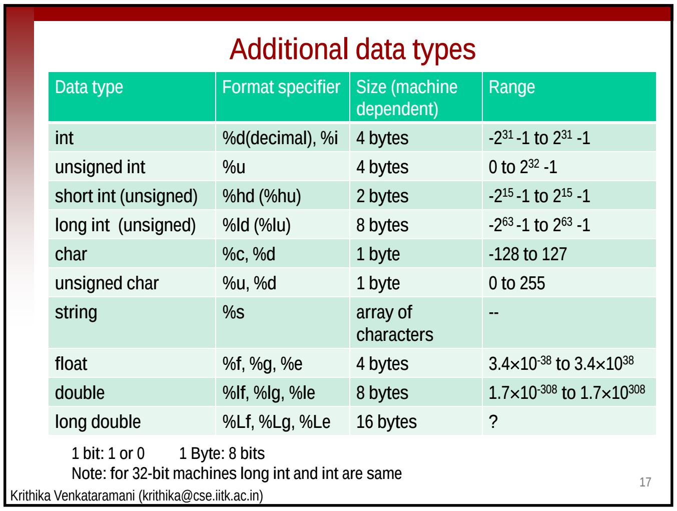
The ASCII Table



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

Printing the Code of a Character

```
/*Program to print the code of a character*/
#include <stdio.h>
void main()
{
    int code;  //Declare variable to store the code
    code = (int) getchar(); //Asking user to input the character
    printf("%d", code); //printing the code of the character
}
```

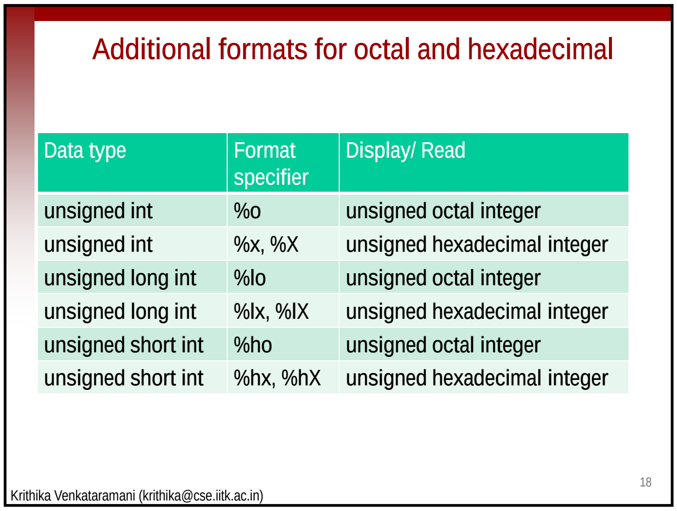


Additional data types

Data type	Format specifier	Size (machine dependent)	Range
int	%d(decimal), %i	4 bytes	$-2^{31}-1$ to $2^{31}-1$
unsigned int	%u	4 bytes	0 to $2^{32}-1$
short int (unsigned)	%hd (%hu)	2 bytes	$-2^{15}-1$ to $2^{15}-1$
long int (unsigned)	%ld (%lu)	8 bytes	$-2^{63}-1$ to $2^{63}-1$
char	%c, %d	1 byte	-128 to 127
unsigned char	%u, %d	1 byte	0 to 255
string	%s	array of characters	--
float	%f, %g, %e	4 bytes	3.4×10^{-38} to 3.4×10^{38}
double	%lf, %lg, %le	8 bytes	1.7×10^{-308} to 1.7×10^{308}
long double	%Lf, %Lg, %Le	16 bytes	?

1 bit: 1 or 0   1 Byte: 8 bits

Note: for 32-bit machines long int and int are same



Additional formats for octal and hexadecimal

Data type	Format specifier	Display/ Read
unsigned int	%o	unsigned octal integer
unsigned int	%x, %X	unsigned hexadecimal integer
unsigned long int	%lo	unsigned octal integer
unsigned long int	%lx, %lX	unsigned hexadecimal integer
unsigned short int	%ho	unsigned octal integer
unsigned short int	%hx, %hX	unsigned hexadecimal integer