4 Data Types and Expressions

- 2 and 2.0 are not the same number
  - A whole number such as 2 is of type **int**
  - A real number such as 2.0 is of type **double**
- Numbers of type **int** are stored as exact values
- Numbers of type **double** may be stored as approximate values due to limitations on number of significant digits that can be represented

4.1 Writing Constants

4.1.1 Writing Integer constants

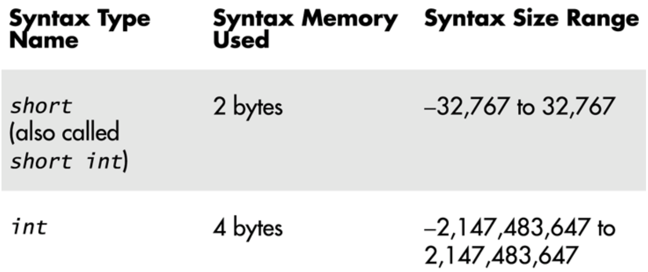
- Type int does not contain decimal points
  - Examples: 34 45 1 89

4.1.2 Writing Double Constants

- Type double can be written in two ways
  - Simple form must include a decimal point
    - Examples: 34.1 23.0034 1.0 89.9
  - Floating Point Notation (Scientific Notation)
    - Examples:
      - 3.41e1 means 34.1
      - 3.67e17 means 367000000000000000.0
      - 5.89e-6 means 0.00000589
  - Number left of **e** does not require a decimal point
  - Exponent cannot contain a decimal point

4.2 Other Number Types

- Various number types have different memory requirements
  - More precision requires more bytes of memory
  - Very large numbers require more bytes of memory
  - Very small numbers require more bytes of memory



Syntax Type Name	Syntax Memory Used	Syntax Size Range
short (also called *short int*)	2 bytes	–32,767 to 32,767
int	4 bytes	–2,147,483,647 to 2,147,483,647

`http://www.cplusplus.com/doc/tutorial/variables/`

4.3 Integer types

- **long** or **long int** (often 4 bytes)
  - Equivalent forms to declare very large integers
    ```
    long big_total;
    long int big_total;
    ```
- **short** or **short int** (often 2 bytes)
  - Equivalent forms to declare smaller integers
    ```
    short small_total;
    short int small_total;
    ```

4.4 Floating point types

- **long double** (often 16 bytes, depends on system)
  - Declares floating point numbers with up to 34 significant digits
    ```
    long double big_number;
    ```
- **float** (often 4 bytes)
  - Declares floating point numbers with up to 7 significant digits
    ```
    float not_so_big_number;
    ```

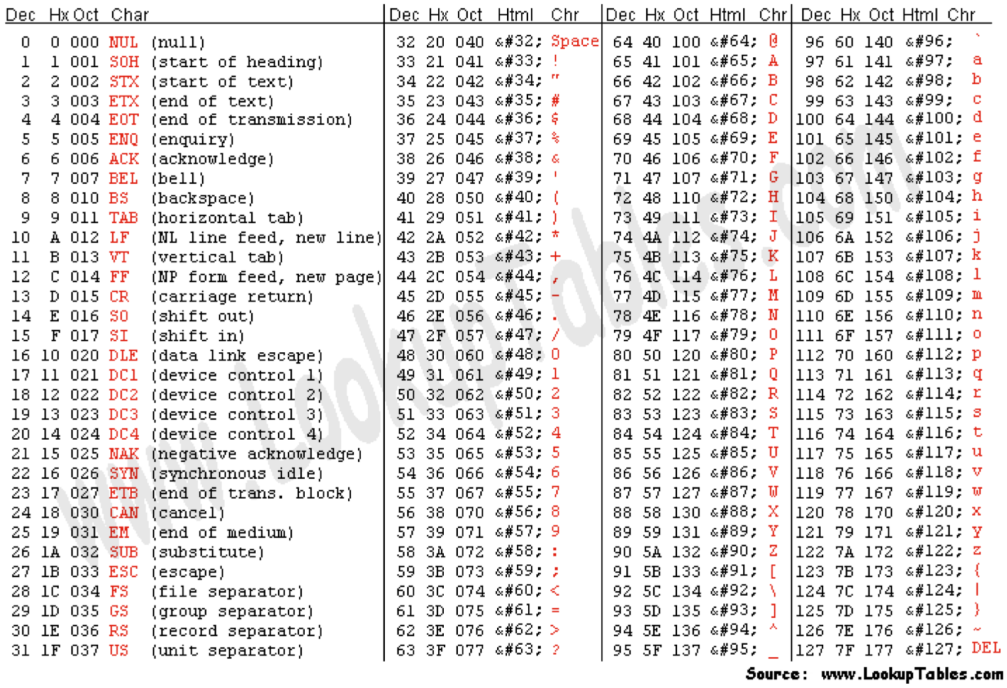
4.5 Type char

- Computers process character data too
- **char**
  - Short for character
  - Can be any single character from the keyboard
- To declare a variable of type **char**:
  ```
  char letter;
  ```

4.5.1 char constants

- Character constants are enclosed in single quotes
  ```
  char letter = 'a';
  ```
- Strings of characters, even if only one character is enclosed in double quotes
  - "a" is a string of characters containing one character

- 'a' is a value of type character



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	&#52;	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	&#123;	{
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	&#125;	}
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

Source: www.LookupTables.com

ASCII Table (American standard code for information interchange)

- This doesn't work. You will get an error!!

```
char letter = "a";
```

because "a" is a string of characters (even just one character). To store a string of characters, you need an array of **char**.

4.5.2 Reading Character Data

- **cin** skips blanks and line breaks looking for data
- The following reads two characters but skips any space that might be between

```
char symbol1, symbol2;
cin >> symbol1 >> symbol2;
```

- User normally separate data items by spaces

J D

- Results are the same if the data is not separated by spaces

JD

4.5.3 sample03

```
/*
 * sample03.cpp
 *
 *  Created on: Jan 15, 2016
 *      Author: fuji
 */

#include <iostream>

int main(int argc, const char * argv[]){
    char a, b;
    std::cout << "input two characters :";
    std::cin >> a >> b;

    std::cout << "You input " << a << " and "<< b << std::endl;

    return 0;
}
```

4.6 Type bool

- **bool** is a new addition to C++
  - Short for boolean
  - Boolean values are either **true** or **false**
- To declare a variable of type bool:
  ```
  bool old_enough;
  ```

4.7 Type Compatibilities

- In general store values in variables of the same type
  - This is a type mismatch:
    ```
    int int_variable;
    int_variable = 2.99;
    ```
  - If your compiler allows this, **int_variable** will most likely contain the value **2**, not **2.99**

4.7.1 int ⟷double

- Variables of type **double** should not be assigned to variables of type **int**
  ```
  int int_variable;
  double double_variable;
  double_variable = 2.00;
  int_variable = double_variable;
  ```
- If allowed, **int_variable** contains **2**, not **2.00**
- Integer values can normally be stored in variables of type **double**
  ```
  double double_variable;
  double_variable = 2;
  int_variable = double_variable;
  ```
- **double_variable** will contain **2.0**

4.7.2 char ⟷int

- The following actions are possible but generally not recommended!
- It is possible to store char values in integer variables

  ```
  int value = 'A';
  ```

  value will contain an integer representing 'A'
- It is possible to store int values in char variables

  ```
  char letter = 65;
  ```

4.7.3 bool ⟷int

- The following actions are possible but generally not recommended!
- Values of type **bool** can be assigned to **int** variables
  - **True** is stored as **1**
  - **False** is stored as **0**
- Values of type **int** can be assigned to **bool** variables
  - Any non-zero integer is stored as **true**
  - Zero is stored as **false**

4.8 Arithmetic

- Arithmetic is performed with operators
  + for addition
  - for subtraction
  * for multiplication
  / for division
- Example: storing a product in the variable **total_weight**

  ```
  total_weight = one_weight * number_of_bars;
  ```

4.8.1 Results of Operators

- Arithmetic operators can be used with any numeric type
- An operand is a number or variable used by the operator
- Result of an operator depends on the types of operands
  - If both operands are **int**, the result is **int**
  - If one or both operands are **double**, the result is **double**

4.8.2 Division of Doubles

- Division with at least one operator of type **double** produces the expected results.

  ```
  double divisor, dividend, quotient;
  divisor = 3;
  dividend = 5;
  quotient = dividend / divisor;
  ```

  - quotient = 1.6666. . .
  - Result is the same if either dividend or divisor is of type **int**

4.8.3 Division of Integers

- Be careful with the division operator!
  - **int** / **int** produces an integer result (true for variables or numeric constants)

    ```
    int dividend, divisor, quotient;
    dividend = 5;
    divisor = 3;
    quotient = dividend / divisor;
    ```

  - The value of quotient is **1, not 1.666. . . (not even 2)**
  - Integer division does not round the result, the fractional part is discarded!

4.8.4 Integer Remainders

- % operator gives the remainder from integer division

  ```
  int dividend, divisor, remainder;
  dividend = 5;
  divisor = 3;
  remainder = dividend % divisor;
  ```

- The value of remainder is 2

4.8.5 sample04

```
/*
 * sample04.cpp
 *
 *  Created on: Jan 15, 2016
 *      Author: fuji
 */
#include <iostream>

int main(int argc, const char * argv[]){
        int dividend, divisor;

        std::cout << "input two integers :\n";
        std::cout << "dividend=";
        std::cin >> dividend;
        std::cout << "divisor=";
        std::cin >> divisor;

        int quotient = dividend / divisor;
        int remainder = dividend % divisor;

        std::cout << dividend << " / " << divisor << " = " << quotient << std::endl;
        std::cout << dividend << " % " << divisor << " = " << remainder << std::endl;
        return 0;
}
```

4.8.6 Type Casting

The problems in 4.8.3. If the variable for quotient declared as **double**,

```
        int dividend, divisor;
        double quotient;
        dividend = 5;
        divisor = 3;
        quotient = dividend / divisor;
```

The value of quotient is still 1, not 1.666… A Type Cast produces a value of one type from another type

- **static_cast<double>(int_variable)**

produces a double representing the integer value of int_variable. For example:

```
        quotient = static_cast<double>(dividend) / divisor;
```

The value of quotient is 1.666…
It also works with old C-style type cast:

```
        quotient = (double)dividend / divisor;
```

4.8.7 Arithmetic Expressions

- Use spacing to make expressions readable
  - Which is easier to read?
    `x+y*z` or `x + y * z`
- Precedence rules for operators are the same as used in your algebra classes
- Use parentheses to alter the order of operations
  `x + y * z` (y is multiplied by z first)

  `(x + y) * z` (x and y are added first)

4.8.8 Operator Shorthand

- Some expressions occur so often that C++ contains to shorthand operators for them
- All arithmetic operators can be used this way
  `count = count + 2;` becomes `count += 2;`

  `bonus = bonus * 2;` becomes `bonus *= 2;`

  `time = time / rush_factor;` becomes `time /= rush_factor;`

  `remainder = remainder % (cnt1+ cnt2);` becomes `remainder %= (cnt1 + cnt2);`

4.9 Increment / Decrement

For a integer variable,

```
        int n = 0;
        n++;
```

'n++' increment 1.
this is equivalent with

```
        int n = 0;
        n += 1;
```

For decrement,

```
        int n = 0;
        n--;
```

4.9.1 n++ or ++n

Those are same, but programs below behave different.

```
        int n = 0;
        std::cout << n++ << srd::endl;
```

```
0
```

```
        int n = 0;
        std::cout << ++n << srd::endl;
```

```
1
```