

Chapter 3: Arithmetic, Variables, Input, Constants, & Library Functions

Mr. Dave Clausen
La Cañada High School

Integer Arithmetic

- + Addition
- - Subtraction
- * Multiplication
- / Quotient (Integer Division)
- % Remainder (Modulus)

$$\text{Divisor} \overline{) \text{Dividend}} \begin{matrix} \text{Quotient} \\ \hline \end{matrix} + \frac{\text{Remainder}}{\text{Divisor}}$$

Integer Order Of Operations

- Expressions within parentheses
 nested parentheses: from inside out
- * (multiplication), % (modulus), / (division)
 from left to right
- + (addition), - (subtraction)
 from left to right

Integer Arithmetic (Examples)

$$(3-4)*5 = -5$$

$$3 * (-2) = -6$$

$$17 / 3 = 5$$

$$17 \% 3 = 2$$

$$17 / (-3) = -5$$

$$-17 \% 7 = -3$$

$$-42+50\%17= -26$$

Integers

- Stored as binary numbers inside the computer.
- Integers produce exact answers
- Int_Min and Int_Max
-2,147,483,648 and 2,147,483,647
- Integer Overflow
 - a number is too large or too small to store
 - no error message
 - unpredictable value

Real Number Arithmetic

- Type double:
- + Addition
- - Subtraction
- * Multiplication
- / Division

Real Number Order Of Operations

- Expressions within parentheses
 nested parentheses: from inside out
- * (multiplication), / (division)
 from left to right
- + (addition), - (subtraction)
 from left to right

Real Number Arithmetic (Examples)

$$2.0 * (1.2 - 4.3) = -6.2$$

$$2.0 * 1.2 - 4.3 = -1.9$$

$$-12.6 / (3.0 + 3.0) = -2.1$$

$$3.1 * 2.0 = 6.2$$

$$-12.6 / 3.0 + 3.0 = -1.2$$

Real Numbers

- Representational errors

precision of data reduced because of the order in which operations are performed

$$(-45.5 + 45.6) + 0.215 = 0.315$$

- $0.1 + 0.215 = 0.315$

$$-45.5 + (45.6 + 0.215) = 0.3$$

- if three digits of accuracy are the computers limit

- $45.6 + 0.215 = 45.815$ or 45.8

- $-45.5 + 45.8 = 0.3$

Real Numbers

- Cancellation Error

lost data due to differences in the precision of operands

$2 + 0.0005 = 2.0005$ but only 2.00 if 3 digits of precision

If possible, add all small numbers before adding to a larger number

Real Overflow: trying to store very large numbers

Real Number Limits

- `DBL_MIN`
 $2.22507e-308$
- `DBL_MAX`
 $1.79769e+308$
- Number of digits in double: 15

Variables

- **Memory Location**
storage cell that can be accessed by address
- **Variable**
memory location, referenced by identifier,
whose value can be changed during a program
- **Constant**
Symbol whose value can't be changed in the
body of the program

Assignment Statements

- A Method of putting values into memory locations
 - $\langle \text{variable name} \rangle = \langle \text{value} \rangle;$
 - $\langle \text{variable name} \rangle = \langle \text{expression} \rangle;$
- Assignment is made from right to left
- Constants can't be on left side of statement
- Expression is a Constant or variable or combination thereof

Assignment Statements

- Values on right side not normally changed
- variable and expression must be of compatible data types (more later)
- Previous value of variable discarded to make room for the new value
- For now, char, int, and double are compatible with each other

Assignment Examples

- `score1 = 72.3;`
- `score2 = 89.4;`
- `score3 = 95.6;`
- `average = (score1 + score2 + score3) / 3.0`
why not divide by 3 instead of 3.0?

Compound Assignments

- “Short hand” notation for frequently used assignments (We will not use these for readability of our programs.)

Short hand

Longer form

$x += y$

$x = x + y$

$x -= y$

$x = x - y$

$x *= y$

$x = x * y$

$x /= y$

$x = x / y$

$x \% = y$

$x = x \% y$

Sample Program

Here is a program that prints data about the cost of three textbooks and calculates the average price of the books:

[Books.cpp](#)

[Books.txt](#)

Software Engineering

- Self-documenting code

Code that is written using descriptive identifiers

- Always use descriptive variable names and constant names

Remember: aim for 8 to 15 characters

Turbo C++ 3.0 for DOS and Turbo C++ 4.5 for Windows both can handle 32 characters for identifier names

Input

- Cin (pronounced see-in)

gets data from keyboard, the standard input stream

extractor operator >>

- obtain input from standard input stream and direct it to a variable (extract from stream to variable)

inserter operator <<

- insert data into standard output stream

EGG ILL

- Extractor Greater Greater, Inserter Less Less

Input

- Data read in from keyboard must match the type of variable used to store data
- Interactive Input
 - enter values from keyboard while the program is running
 - cin causes the program to stop and wait for the user to enter data from the keyboard
 - prompt the user for the data (user friendly)

Input: Sample Programs

No prompt for any of the data values:

[input.cpp](#)

[input.txt](#)

One prompt for each data value (preferred)

[triples.cpp](#)

[triples.txt](#)

Character Data

- Type char
 - each char is associated with an integer value
- Collating sequence
 - order of character data used by the computer
- Character set
 - the character list available
 - ASCII (American Standard Code for Information Interchange) on our systems: page 85

ASCII Code

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	"	#	\$	%	&	`
4	()	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[\]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~	DEL		

Full ASCII Code Chart

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		DEL

Source: www.LookupTables.com

Constants

- Symbolic constant: `PI`
- Literal constant: `3.14`
- Constant declaration section
after Preprocessor Directives
before type definition section, program heading,
and the `int main()` function.
Literal constant is assigned to symbolic constant
 - `const double PI = 3.14;`
- Style for constants is `ALL_CAPS`

Rationale for using Constants

- Programs are easier to read
- Easier to change values that are currently fixed but subject to change in the future
i.e. `STATE_TAX_RATE`
change one line, rather than searching every line
- Programs more reliable
fewer chances for typos
compiler will “catch” named constant typos

Library Constants

- What are the ranges from minimum to maximum for the types char, int, double, etc.?
 - # include <limits.h>;
 - # include <float.h>;
- Varies by computer system
- Here is a test program to display the values

[sizes.cpp](#)

[sizes.txt](#)

String Variables

- apstring data type

```
#include "apstring.cpp";
```

- note quotes “”, not angle brackets < >
- not included with compilers, download and install

String Input with >>

- The >> operator ignores leading whitespace space, tab, or carriage return
- Then reads nonblank characters until next whitespace character user is allowed to use backspace or delete
 - until next whitespace character
- Upon return or whitespace string is stored
- >> Can't be used for strings with spaces.

Page 88 [Example 1.ide](#)

String Input with getline

- Getline function

reads characters, tab, space into string variable
until newline ('\n') char

the newline char is not stored in the string
variable

```
getline (<input stream>, <string variable>);
```

- **getline (cin, name);**

doesn't ignore leading whitespace characters

[Pg88ex2.ide](#)

Using cin>> before getline

- >> reads and stores up to newline
- getline reads newline as first char of line
- and quits reading at newline
- any string variables in getline are empty

[Pg89ex3.ide](#)

Solutions for >> before getline

- Use `getline (cin, consume_newline)` to consume newline character
- This is the required method, for example:

[P89ex5.cpp](#)

More apstring

- Length function returns number of characters in a string

`<string variable>.length()`

- `cout <<“Length of “ <<word <<” = “ <<word.length() <<endl;`
- `length_of_word = word.length();`
- No memory is allocated when a string variable is declared, length is zero characters.
- Empty string is “”
length is zero characters.

apstring Member Functions

apstring Member Function	What It Does	Example Use
<code>int length()</code>	Returns the number of characters in the string.	<pre>apstring word = "Hello there"; int word_length = word.length(); cout<<word_length; // Displays 11</pre>
<code>int find(<a string>)</code>	Returns the starting position of the first occurrence of a string or -1 if the string does not exist.	<pre>apstring word = "Hello there"; int location = word.find("there"); cout << location; // Displays 6</pre>
<code>int find(<a character>)</code>	Returns the starting position of the first occurrence of a character or -1 if the character does not exist.	<pre>apstring word = "Hello there"; int location = word.find('H'); cout << location; // Displays 0</pre>
<code>apstring substr(<position>, <length>)</code>	Returns a substring of length characters starting at position	<pre>apstring word = "Hello there"; apstring word2 = word.substr(3, 2); cout << word2; // Displays "lo"</pre>

String Concatenation

- Concatenation

an operation to append the contents of one data structure after the contents of another data structure

+ means concatenate for strings

+ means addition for numbers

Concatenation Example 1

- To create a new string
apstring first, second, third;

```
first = "Hi";
```

```
second = " there";
```

```
third = first + second;
```

```
cout<<third;
```

Concatenation Example 2

- To append a character to the end of a string

```
apstring singular; //error in textbook, this is correct
```

```
singular = "fish";
```

```
cout << singular + "es";
```

Library Functions

- Different versions of C++ have different library functions available.
- Form for using a function:
 <function name> (<argument list>);
 an argument is an expression, variable or constant
- A function is invoked or called when used in a statement
 - `answer = pow(3,4);`
 - `result = pow(base, exponent);`

Library Functions

- List of Library Functions in Appendix 2
- Two to know for now...

sqrt square root

pow raise a base to a power

Examples:

- `sqrt (25)` `sqrt (25.0)`
- `pow(2,4)` `pow(-3,5)` `pow(2.0,4)`
- `square_root = sqrt (number);`
- `answer = pow(base, exponent);`

Sample Library Functions

Function Declaration

```
double fmod(double x, double y);  
double log(double x);  
double pow(double x, double y);  
double sqrt(double x);  
double cos(double x);
```

Action of Function

returns floating-point remainder of x / y
returns natural logarithm of x
returns x raised to power of y
returns square root of x
returns cosine of x

Expression	Value
<code>pow(2, 4)</code>	16
<code>pow(2.0, 4)</code>	16.0
<code>pow(-3, 2)</code>	9
<code>fmod(5.3, 2.1)</code>	1.1
<code>sqrt(25.0)</code>	5.0
<code>sqrt(25)</code>	5
<code>sqrt(0.0)</code>	0.0
<code>sqrt(-2.0)</code>	Not permissible

Member Functions

- Some library functions are associated with a data type called classes.
- Class: a description of the attributes and behavior of a set of computational objects.
- Member function: an operation defined for a class of objects
- Member functions are called using a different syntax.

Member Function Syntax

- Conventional functions
 <conventional function name> (variable name)
- Member functions
 <variable name>.<member function name> ()
 for example:
 - cout<< word.length()
 - number_of_characters = word.length()

Declaring Variables & Constants

Examples

```
const double TAX_RATE = 0.75;
```

```
int main( )  
{  
    int sum, counter;  
    int total = 0, product = 1;  
    double average;  
    char repeat_program;  
    apstring name;
```

Type Compatibility

- Mixed mode expressions
 - expressions with different data types
 - int, char, double, etc. in the same expression
- Pascal and BASIC would give a Type Mismatch Error Message and quit
- C++ does not give any error messages for this
- Extra care is necessary for mixed mode expressions

Type Conversion

- Type promotion

converting a less inclusive data type into a more inclusive data type (i.e. int to double)

When adding an integer to a double, the compiler converts the integer to type double, adds, and gives an answer of type double.

int, char, and double are “compatible”

Implicit Type Conversions

```
int_var = double_var;  
double_var = int_var;  
int_var = char_var;  
char_var = int_var;
```

Ex.

```
whole_num='A' + 1;  
digit = '5' - '0';
```

Truncates the decimals
adds .0

get ASCII code of char
get the character whose
ASCII code is the
integer value

66

5

Case Changing of Character Values

```
lower_case = upper_case - 'A' + 'a';
```

OR:

```
const int CASE_CHANGE = 32;
```

```
lower_case = upper_case + CASE_CHANGE;
```

```
upper_case = lower_case - CASE_CHANGE;
```

OR: #include <ctype.h>;

```
lower_case = tolower (upper_case);
```

```
upper_case = toupper (lower_case);
```

Case Changing of Character Values 2

```
integer_variable = character_variable;
```

```
int_var = int (char_var);
```

Type Casts

- Type cast
 - an operation that a programmer can use to convert the data type
- Explicit type conversion
 - the use of an operation by the programmer to convert one type of data into another
- Form of type cast
 - `<type name> (<expression>);`
 - `(<type name>) <expression>;`

Explicit Type Conversion Examples

```
cout<< int (double_variable) <<endl;
```

```
cout<< (long int) integer_var;
```

```
answer = double (numerator) / double (denominator);
```

Type casting can add clarity to your program while reminding you of the data types involved in your calculations.

Explicit Type Conversion

Examples 2

//cast a double to an int: loses decimals

```
int_var = (int) double_var;
```

//cast an int to a double: adds .0

```
double_var = (double) int_var;
```

//cast a char to an int: get ASCII code of char

```
int_var = (int) char_var;
```

//cast an int to a char: get the character if in range

```
char_var = (char) int_var;
```