C++ Programming

Instructor: Rita Kuo
Office: CS 520E
Phone: Ext. 4405
E-mail: rita.kuo@uvu.edu

Mapping zyBooks Chapters

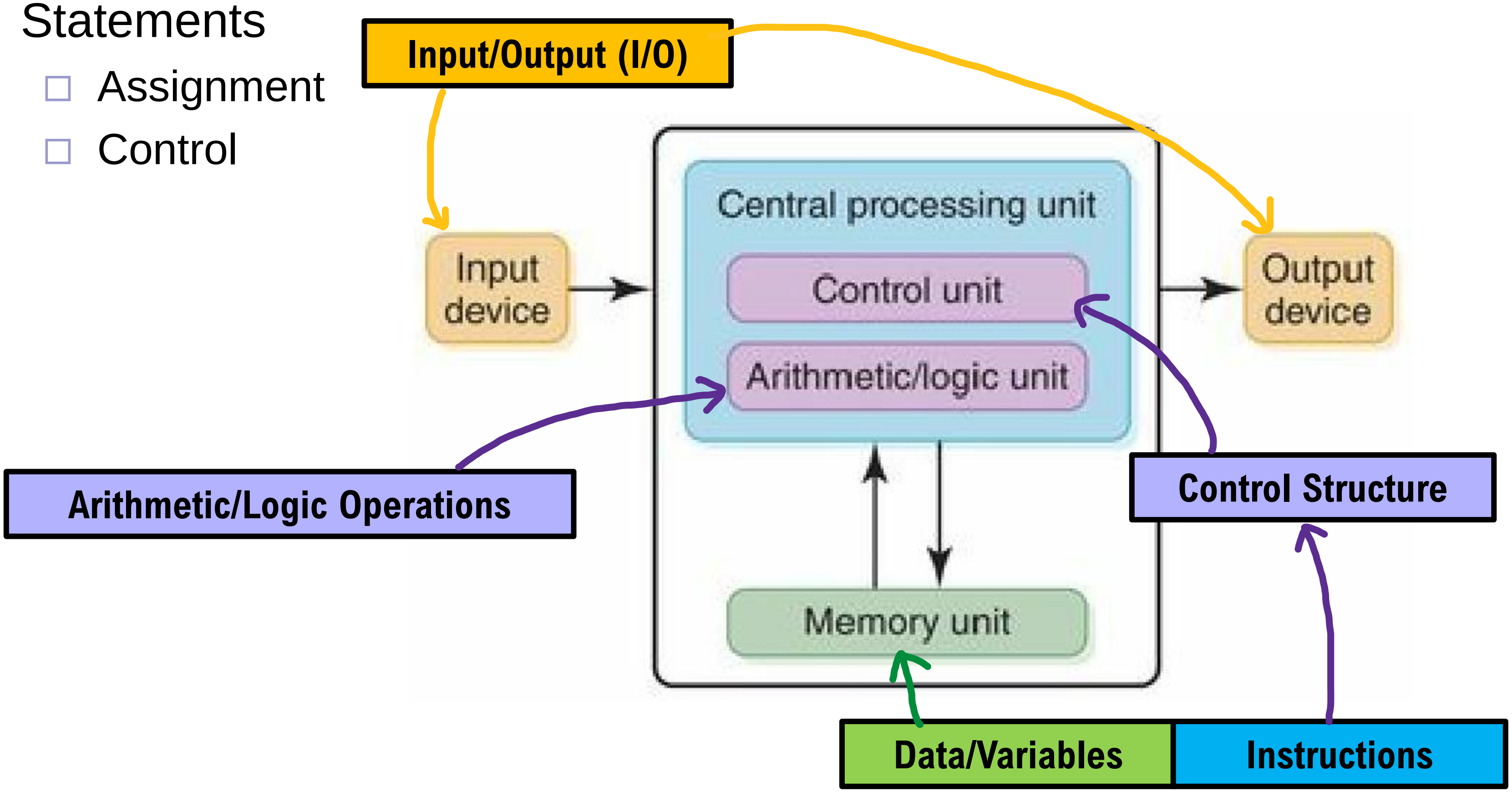
Topics on the slides	Chapters in zyBooks
Variables and Assignments	2.1, 2.2, 2.3
Variable Types	
Floating-Point Variables	2.7, 2.8, 2.18, 2.19
Characters	2.15
Integer Data	2.18, 2.20
Integer Overflow	2.17
Integer Division and Modulo	2.11
Constant Variables	2.9
Arithmetic Operators	2.4
Compound Operators	2.5
Math Library	2.10
Auto Data Type (Since C++ 11)	2.12
Type Conversions	2.13
String	2.16

Not in the slides but should be covered in the pre-req courses:
2.14 (Please check the instructor note)

Variables and Assignments

Review - Elements in Programming Language

- Input/Output (I/O)
- Variables
- Expression
- Statements
  - □ Assignment
  - □ Control



Input/Output (I/O)

Arithmetic/Logic Operations

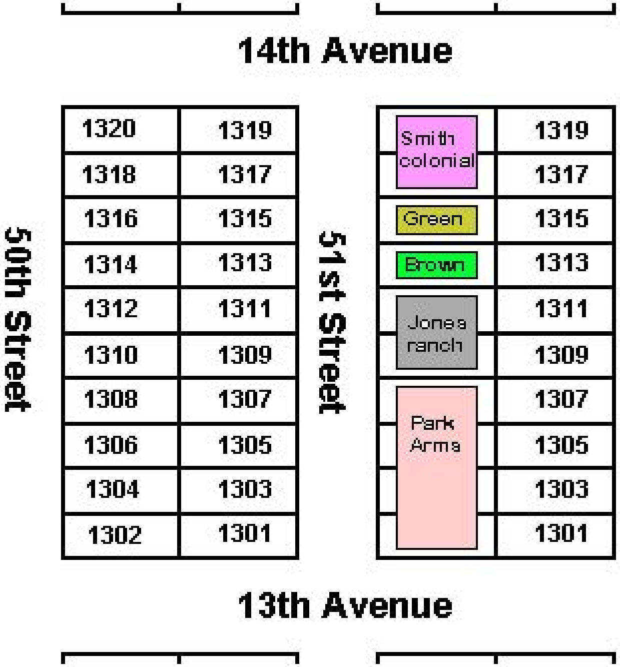
Control Structure

Data/Variables

Instructions

Review - Variables

- A storage area in memory and its symbolic name
- The storage area contains a value that is referenced via the symbolic name
- Being a variable, the value stored in memory can change



- Example: http://www.bernstein-plus-sons.com/.dowling/Prog_Lang_Module/Computer_Memory.html

Review - Assignments

- Assignments
  - Ties the storage area and the symbolic name together
  - Example:

    ```
    x ← 6
    ```

    place 6 in memory and assigns it the name x.
    - **x** has the value of **6**
    - **6** is assigned to **x**
- Assignment in C/C++
  - General pattern for assignment:

    symbolic_name = value;
  - A variable can be given a value by means of assignment
  - Example:

    ```
    x = 6;
    ```
  - **=**: assignment operator, **not** equality in mathematics.
  - **6**: a constant

Review - Assignments

- Assignment in C/C++
  - Assignments do **not** commute. This is wrong:

    `6 ← x` or `6 = x;`

    - Symbolic names (aka identifiers) must begin with a letter or underscore
    - **=**: the value in the right-hand side (RHS) will be assignment to the memory address in the left-hand side (LHS)
      → the program cannot hard-code addresses as memory is ultimately managed by the operating system.
  - The following statements are valid:

    ```
    int x, y;
    y = 3;
    x = y;
    ```

    - In the first statement, **y** is on the LHS of the assignment operator and is the symbolic name
    - In the second statement, **y** is on the RHS of the assignment operator and the value stored at **y** is used
      → the variable **y** is evaluated when it is on the RHS

Expression

- Expression in mathematics
  - A combination of numbers (constants), variables, operations, functions etc.
  - Example:

$2 + 3$

$8x - 5$

$$f(a) + \sum_{k=1}^{n} \frac{1}{k!} \frac{d^k}{dt^k}\Big|_{t=0} f(u(t)) + \int_0^1 \frac{(1-t)^n}{n!} \frac{d^{n+1}}{dt^{n+1}} f(u(t))\, dt.$$

- Expression
  - A combination of values, variables, operators, and functions that return a value
  - Example:

```
2 + x
3 * x * y - 7 * strlen(s)
```

Assignment

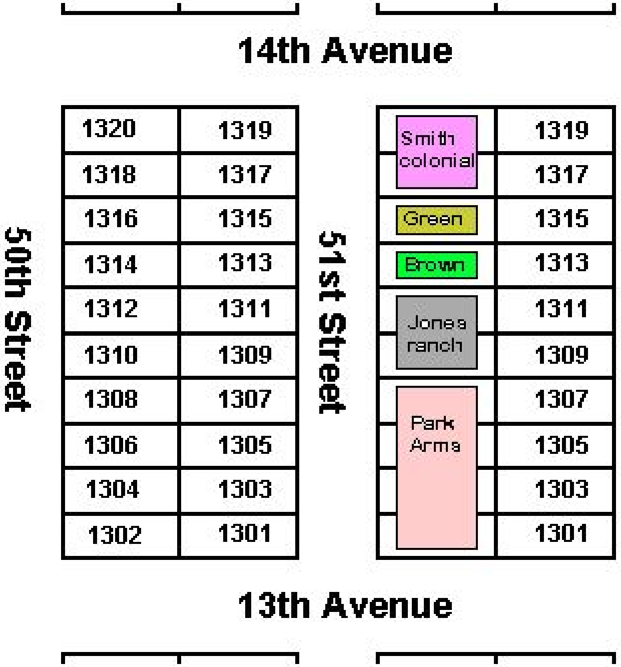
- Assignment Operator
  - **=**: assignment operator, **not** equality in math.
  - **Lvalues**: an object stored in computer memory, not a constant or the result of a computation
  - Most C operators allow their operands to be variables, constants, or expressions containing other operators
  - The assignment operator requires an **lvalue** as its left operand
  - Incorrect statements:

```
12 = i;
i + j = 0;
-i = j;
```

Variable Types

Review - Variables

- A storage area in memory and its symbolic name
- The storage area contains a value that is referenced via the symbolic name
- Being a variable, the value stored in memory can change



- Example:
  http://www.bernstein-plus-sons.com/.dowling/Prog_Lang_Module/Computer_Memory.html

Review - Variables

- Types
  - Specify what kind of data it will hold
  - Basic data types are **integers** (`short`, `int`, `long`), **real numbers** (`float` or `double`), or **characters** (`char`).
  - `int`: hold integer values, i.e., whole numbers such as 7, -11, 0, etc.
    - The largest `int` value is typically 2,147,483,647 but can be as small as 32,767
  - `float`: can store numbers with digits after the decimal point, such as 379.125
    - Slower than `int` in arithmetic operation
    - Is often an approximation of the number. E.g., `0.1` in a float variable might be 0.09999999999999987 stored in the system.
  - **Variables must be declared before they can be used**

Review - Variables

- Declaration
  - Announce the properties of variables
  - Only need to declare variable's type once. Once declared it is immutable
  - Consist of a type name and a list of variables
  - Example:

    ```
    int sum;
    int fahr, celsius;
    ```

  - Because the variables must be declared first, the simple C program form can be rewritten as

```
directives

int main()
{
      declarations
      statements
}
```

Floating-Point Variables

- A real number containing a decimal point that can appear anywhere (or “float”) in the number
  - Example: 98.6, 0.0001,or -55.667.
- Format in Base-10 Numbers
  - $sign \times mantissa \times 10^{exponent}$
  - *sign* : positive or negative
  - *mantissa* : the value with the radix point assumed to be to the right
  - *exponent*: how the radix point is shifted relative to the mantissa
  - Example: $148.69 = +\ 14869 \times 10^{-2}$
- Scientific Notation
  - Decimal point is kept to the right of the leftmost digit
  - Example: 148.69 $= +\ 1.4869 \times 10^{+2}$

    = + 1.4869E+2

Floating-Point Variables

- Scientific Notation
  - Decimal point is kept to the right of the leftmost digit
  - Example: 148.69 $= +\ 1.4869 \times 10^{+2}$
    $= +\ 1.4869\text{E}{+2}$
- Binary Floating-Points
  - $20.25_{10} = 10100.01_2 = +\ 1.010001 \times 2^4$
  - How to save the data in the memory? If we use 2 bytes to save the data:

$$\text{sign} \nearrow +\ 1.\underbrace{010001}_{\text{fraction/mantissa}} \times 2^{\boxed{4}} \leftarrow \text{exponent}$$

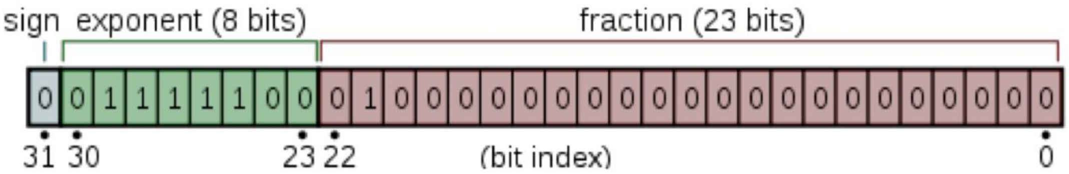
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	exponent					fraction									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0

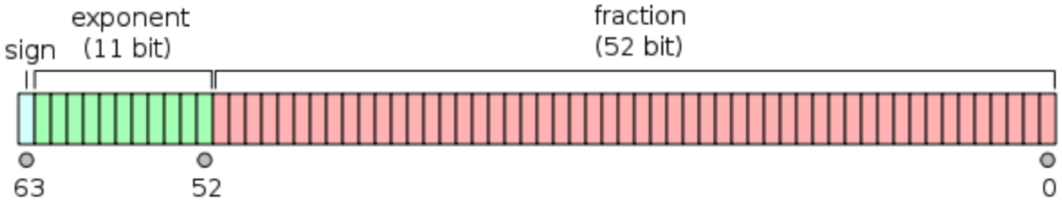
  - Video: https://www.youtube.com/watch?v=KkFLnnneZ2k

Floating-Point Variables

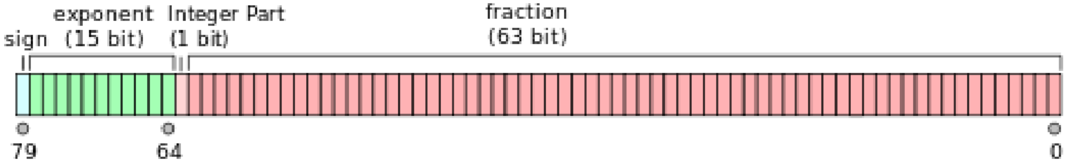
- IEEE-754 Standard
  - Single Precision: stored in 32 bits, where the bits are
    bit 31 = sign, bits 23 - 30 = exponent (8 bits), bits 0 - 22 = mantissa (23 bits)



  - Double Precision: stored in 64 bits, where the bits are
    bit 64 = sign, bits 52 - 63 = exponent (11 bits), bits 0 - 51 = mantissa (52 bits)



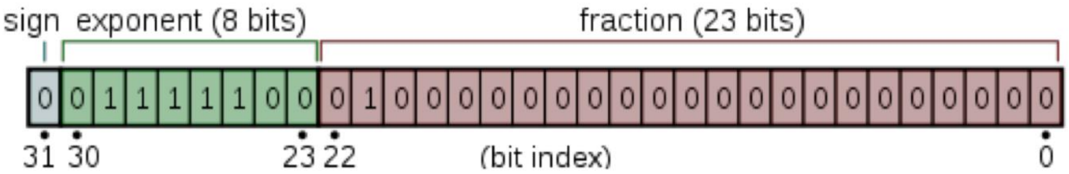
  - Extended Precision: stored in 80 bits, where the bits are
    bit 79 = sign, bits 64 - 78 = exponent (15 bits), bits 0 - 63 = mantissa (64 bits)



    But 80 is not a power of two; the actual size of an extended precision float is 12 (96 bit) or 16 (128 bit) bytes

Floating-Point Variables

- IEEE-754 Standard
  - Single Precision: stored in 32 bits, where the bits are
    bit 31 = sign, bits 23 - 30 = exponent (8 bits), bits 0 - 22 = mantissa (23 bits)



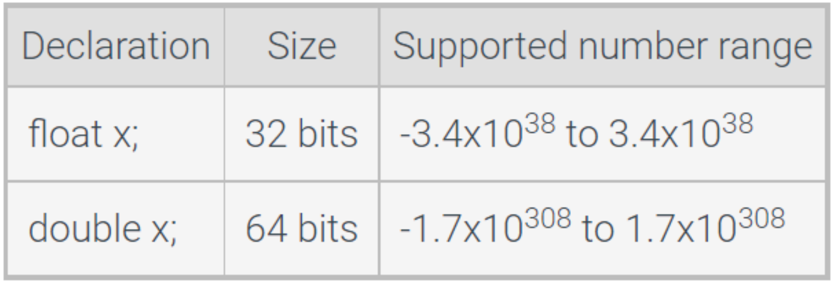
  - Mantissa is an unsigned binary number with bit 23 being 2^{-1} position, bit 22 being 2^{-2} position, etc.
  - The exponent is stored in excess (offset) format to allow for negative exponents. Excess 127 for single precision, 1023 for double, and 16383 for extended.
  - (Reference: https://blog.angularindepth.com/the-mechanics-behind-exponent-bias-in-floating-point-9b3185083528)
  - The calculation for non-zero floating point number is:

$$-1^{sign} \times 2^{exponent\ -\ excess} \times 1.mantissa_{base2}$$

Floating-Point Variables

- IEEE-754 Standard
  - Value Range
    - For single precision:
      min = 1.175494E-38
      max = 3.402923E+38
    - For double precision:
      min = 2.225074E-308
      max = 1.797693E+308
    - For extended precision:
      min = 3.362103E-4932
      max = 1.189731E+4932
  - There is more to the standard than just data representation. How numbers are rounded, errors (NaN) and how positive and negative infinite are handled
    (Reference: https://steve.hollasch.net/cgindex/coding/ieeefloat.html)
- Video: https://www.youtube.com/watch?v=50ZYcZebIec

Floating-Point Variables

- Floating-point variables in C++
  - 

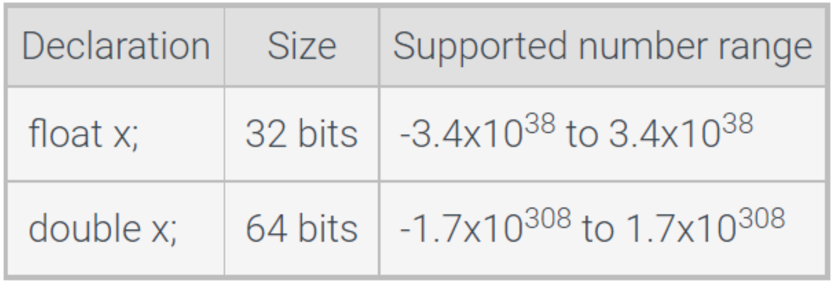
Declaration	Size	Supported number range
float x;	32 bits	-3.4x10^{38} to 3.4x10^{38}
double x;	64 bits	-1.7x10^{308} to 1.7x10^{308}

- Scientific notation for floating-point literals
  - A floating-point literal using scientific notation is written using an e preceding the power-of-10 exponent
  - Example:

```
double avogadrosNumber = 6.02e23;
double G = 6.673e-11;
```

Floating-Point Variables

- Floating-Point Variables in C/C++
  - □



Declaration	Size	Supported number range
float x;	32 bits	-3.4×10^{38} to 3.4×10^{38}
double x;	64 bits	-1.7×10^{308} to 1.7×10^{308}

Check the example in Figure 2.19.1

- Choosing a variable type (**`double`** vs. **`int`**)
  - □ Integer variables are typically used for values that are counted, like 42 cars, 10 pizzas, or -95 days.
  - □ Floating-point variables are typically used for measurements, like 98.6 degrees, 0.00001 meters, or -55.667 degrees.
  - □ Floating-point variables are also used when dealing with fractions of countable items, such as the average number of cars per household
- Inaccurate in floating-point data
  - □ https://www.baeldung.com/cs/floating-point-numbers-inaccuracy

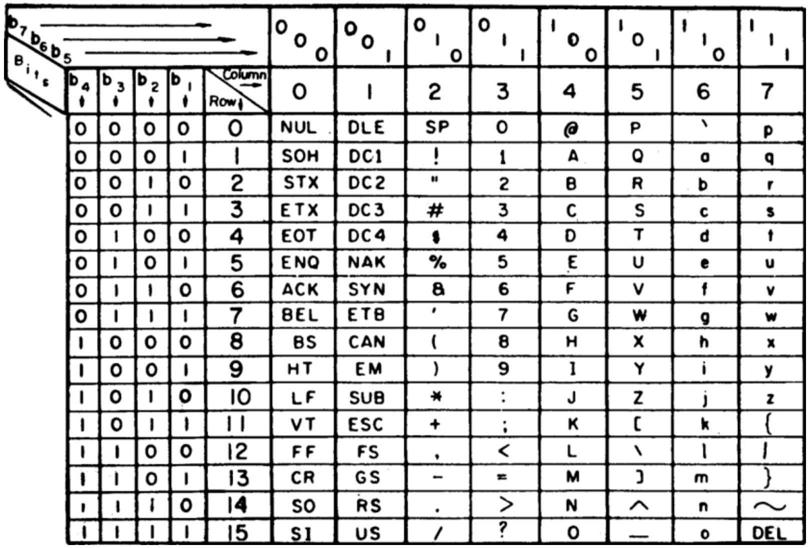
Characters

- Text Representation in Computer System
  - List all characters and assign a binary string to each character
  - Character Set: a list of the characters and the codes used to represent each one
- ASCII Character Set
  - American Standard Code for Information Interchange
  - Developed by Bell Telephone
  - A character-encoding scheme
    - Originally based on the English alphabet
    - Consists of a code that pairs each character from a given set into something else. Typically a bit pattern.
    - A code is a rule to map information (a character) into another representation
    - Examples of codes: Morse code, Braille
  - Maps English characters to bit pattern
  - Consists of 128 characters (33 control characters and 95 printable characters)

Characters

- ASCII Character Set
  - American Standard Code for Information Interchange
  - 7 bits, 128 unique characters

USASCII code chart



Bits b7 b6 b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

  - Examples
    - **A**: `0011 0101` (65 decimal; 0x41 hex)
    - **a**: `0110 0001` (97 decimal; 0x61 hex)

Characters

- Characters in C/C++
  - Declared as a `char` type.
  - Example:

    ```
    char myChar;
    ```

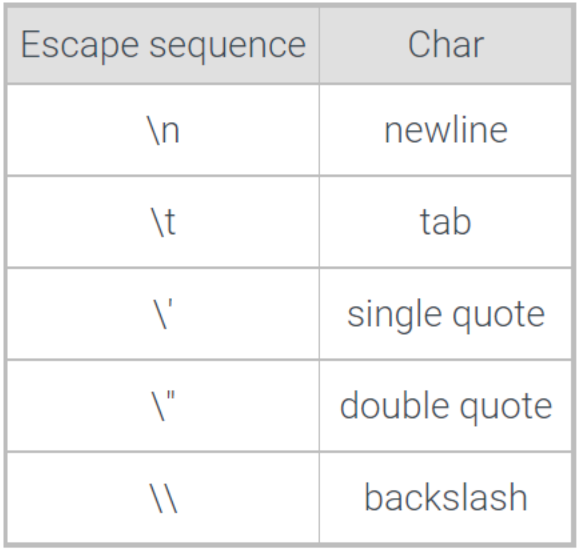
  - Assign a character to a char variable: use single quotes
  - Example:

    ```
    myChar = 'm';
    ```

  - Single quote has different usage than double quote

Escape Sequences

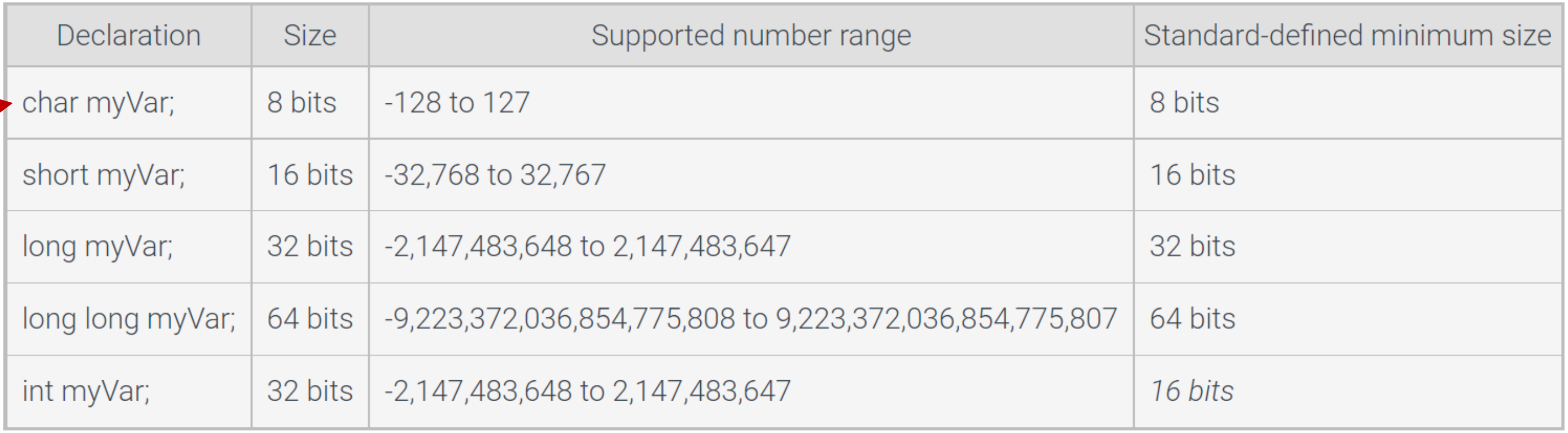
- Special characters encoding in ASCII but no visible character exists
  - E.g., newline, tab, etc
- A two-character sequence starting with \ creates an escape sequence
  - Example



Escape sequence	Char
\n	newline
\t	tab
\'	single quote
\"	double quote
\\	backslash

Integer Data

- Integer Data in C/C++

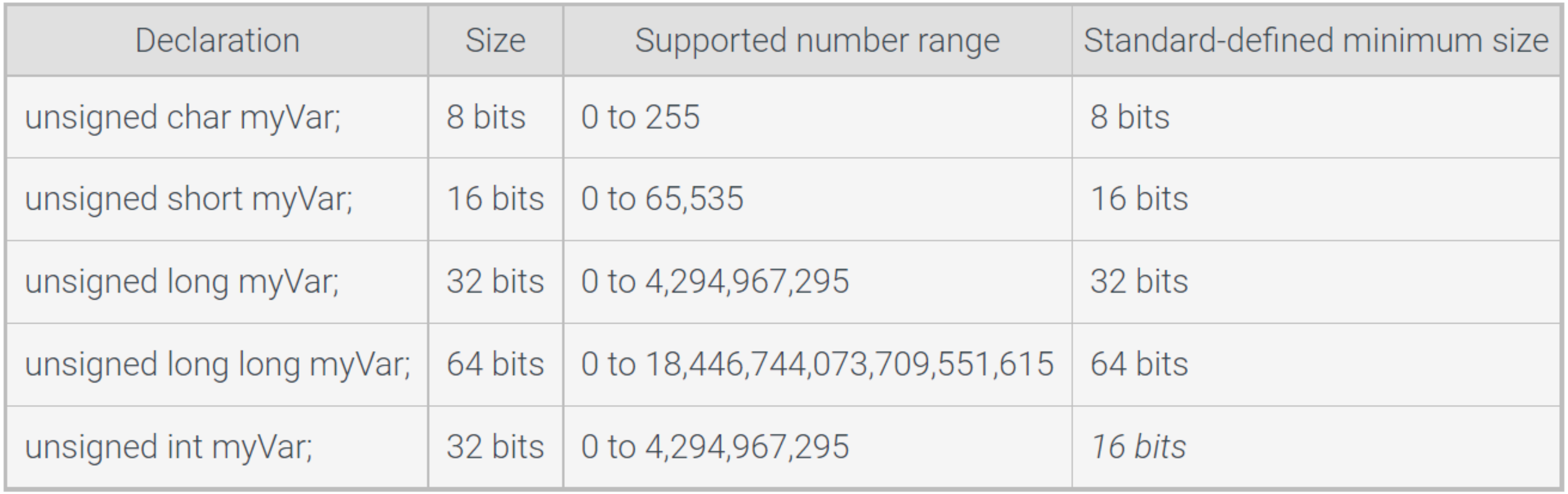


Declaration	Size	Supported number range	Standard-defined minimum size
char myVar;	8 bits	-128 to 127	8 bits
short myVar;	16 bits	-32,768 to 32,767	16 bits
long myVar;	32 bits	-2,147,483,648 to 2,147,483,647	32 bits
long long myVar;	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	64 bits
int myVar;	32 bits	-2,147,483,648 to 2,147,483,647	*16 bits*

- How to store a negative number?
  - □ Signed-Magnitude Representation
  - □ One’s Complement
  - □ Two’s Complement
- https://www.youtube.com/watch?v=Z3mswCN2FJs

Integer Data

- Unsigned Integers

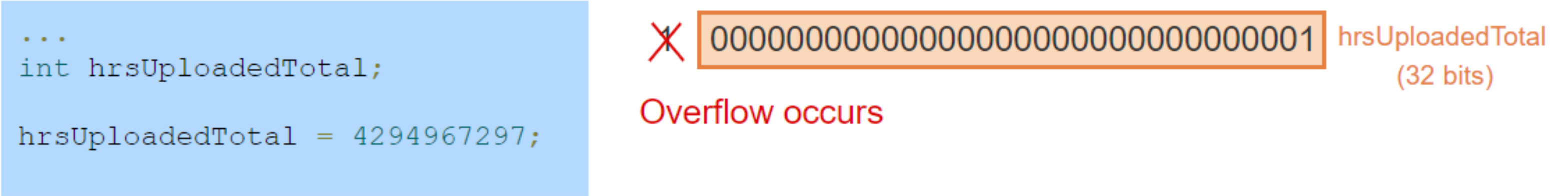


Declaration	Size	Supported number range	Standard-defined minimum size
unsigned char myVar;	8 bits	0 to 255	8 bits
unsigned short myVar;	16 bits	0 to 65,535	16 bits
unsigned long myVar;	32 bits	0 to 4,294,967,295	32 bits
unsigned long long myVar;	64 bits	0 to 18,446,744,073,709,551,615	64 bits
unsigned int myVar;	32 bits	0 to 4,294,967,295	*16 bits*

Participation Activity 2.17.1

Integer Overflow

- Definition
  - Occurs when the value being assigned to a variable is greater than the maximum value the variable can store.
  - Example:

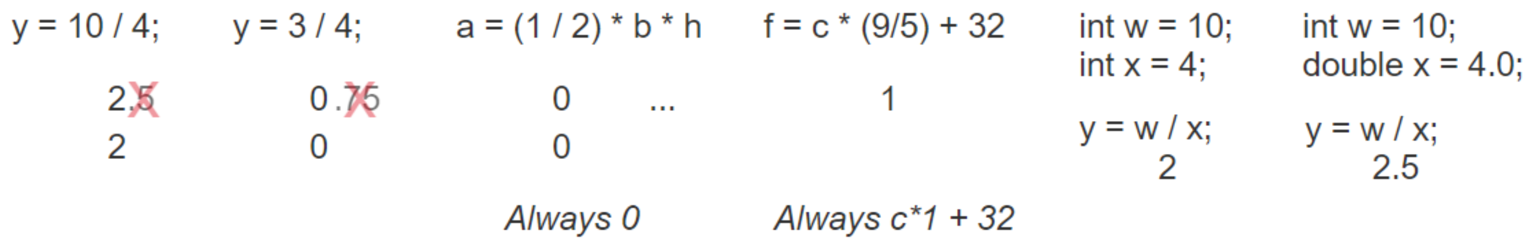


```
...
int hrsUploadedTotal;

hrsUploadedTotal = 4294967297;
```

- Integer Overflow for Signed and Unsigned Integers
  - Signed `int`s become negative, and unsigned `int`s wrap around to 0 when they overflow

Integer Division and Modulo

- Integer Division
  - □ When both operands of the division operator (**/**) are integers, the operator performs integer division, which does not generate any fraction.
  - □ Example



y = 10 / 4;
2.~~5~~
2

y = 3 / 4;
0.~~75~~
0

a = (1 / 2) * b * h
0 ...
0
Always 0

f = c * (9/5) + 32
1
*Always c*1 + 32*

int w = 10;
int x = 4;
y = w / x;
2

int w = 10;
double x = 4.0;
y = w / x;
2.5

  - □ Solution:
    - Convert one of the operand to a floating point data
      Example:
      `y = 10 / 4;` is modified as `y = 10.0 / 4.0;`
    - Use casting in at least one of the operand to convert it to a floating point data

Integer Division and Modulo

- Modulo
  - `a % b` is the remainder after division `a / b`
  - Only works on integral types
  - Examples:

```
4 % 5 = 4
5 % 5 = 0
6 % 5 = 1
```

Digit Separator

- Numeric literals of more than a few digits are hard to read. E.g.,
  - Pronounce 7237498123.
  - Compare 237498123 with 237499123 for equality.
  - Decide whether 237499123 or 20249472 is larger.
- C++14 define Simple Quotation Mark `'` as a digit separator, in numbers and user-defined literals. E.g.,
  - `long long decn = 1'000'000'000ll;`
  - `long long hexn = 0xFFFF'FFFFll;`
  - `long long octn = 00'23'00ll;`
  - `long long binn = 0b1010'0011ll;`
- Single quotes mark are ignored when determining its value

https://riptutorial.com/cplusplus/example/31798/digit-separator

Constant Variables

- A constant is a value or an identifier whose value cannot be altered in a program.
- Declare a constant variable: use `const` keyword
- Examples:

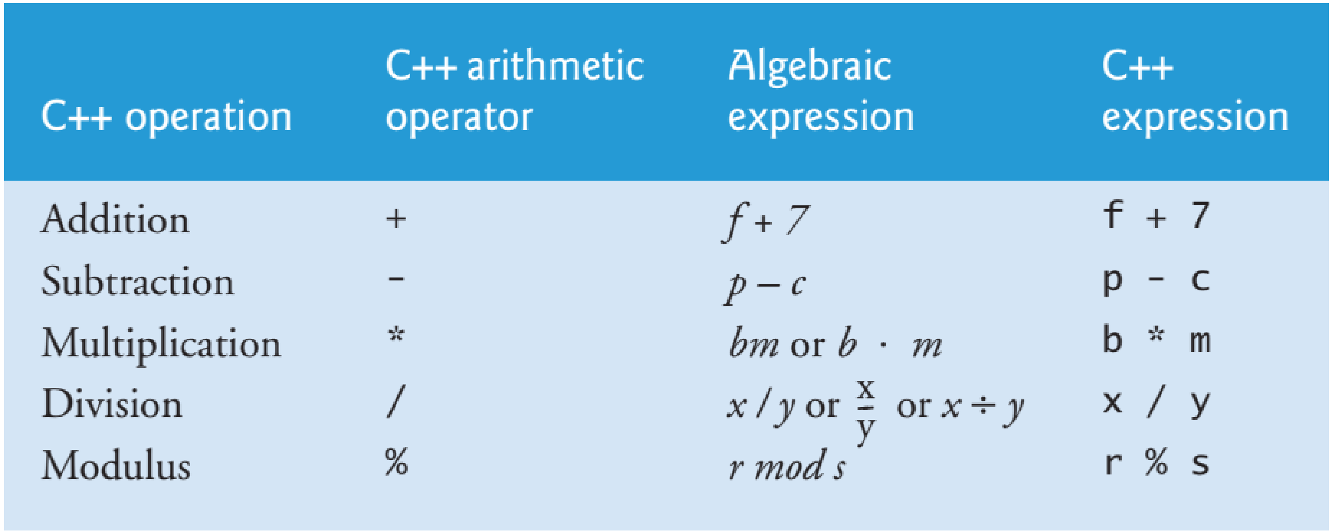
```
const double SPEED_OF_SOUND   = 761.207;
const double SECONDS_PER_HOUR = 3600.0;
```

- A common convention is to name constant variables using upper case letters with words separated by underscores.

Arithmetic Operators

Assignment and Expression

- Arithmetic Operators
  - □ Arithmetic Operators in C++:

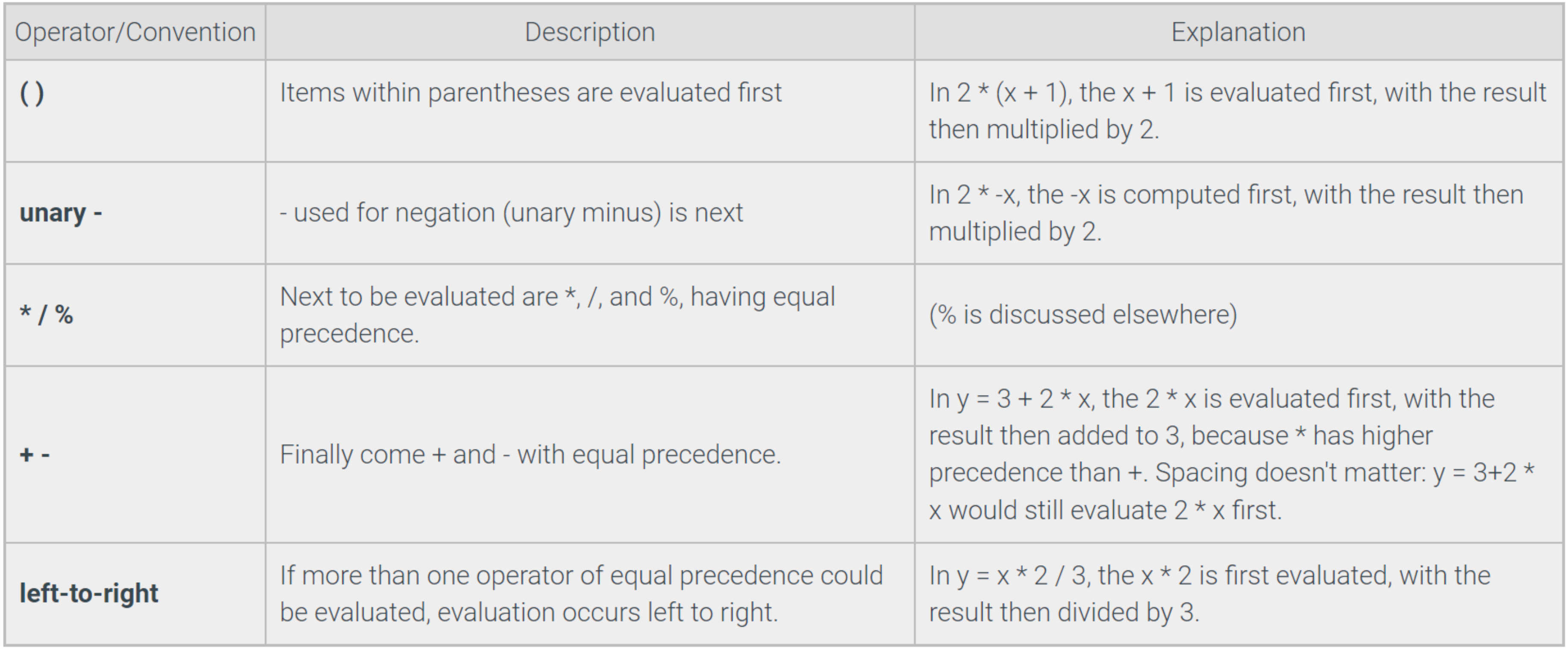


C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	f + 7
Subtraction	-	$p - c$	p - c
Multiplication	*	bm or $b \cdot m$	b * m
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	x / y
Modulus	%	$r \ mod \ s$	r % s

Table 2.4.2

Assignment and Expression

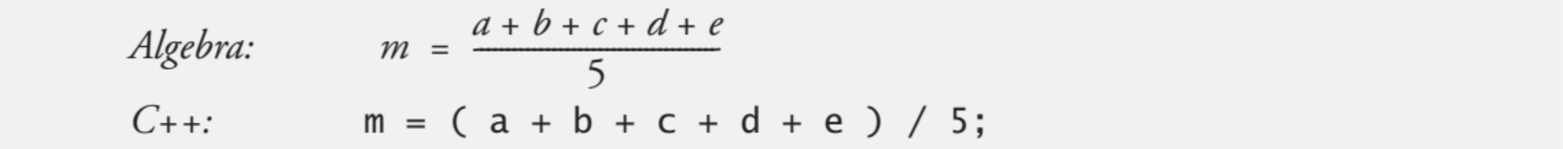
- Arithmetic Operators
  - □ Precedence of Arithmetic Operators in C++:



Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In 2 * (x + 1), the x + 1 is evaluated first, with the result then multiplied by 2.
unary -	- used for negation (unary minus) is next	In 2 * -x, the -x is computed first, with the result then multiplied by 2.
*** / %**	Next to be evaluated are *, /, and %, having equal precedence.	(% is discussed elsewhere)
+ -	Finally come + and - with equal precedence.	In y = 3 + 2 * x, the 2 * x is evaluated first, with the result then added to 3, because * has higher precedence than +. Spacing doesn't matter: y = 3+2 * x would still evaluate 2 * x first.
left-to-right	If more than one operator of equal precedence could be evaluated, evaluation occurs left to right.	In y = x * 2 / 3, the x * 2 is first evaluated, with the result then divided by 3.

Assignment and Expression

- Rules of operator precedence
  - Example

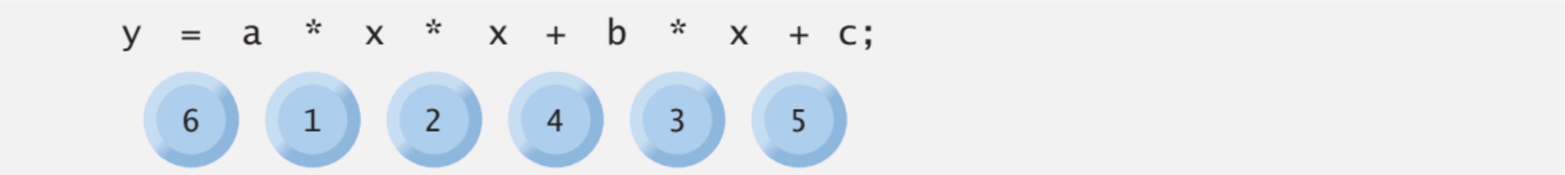


Algebra: $m = \dfrac{a + b + c + d + e}{5}$

C++:
```
m = ( a + b + c + d + e ) / 5;
```

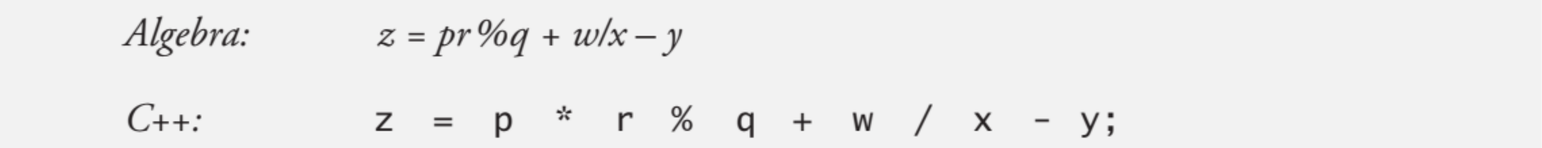
- Incorrect solution: `m = a + b + c + d + e / 5;`

  - Example



```
y = a * x * x + b * x + c;
```

  - Example

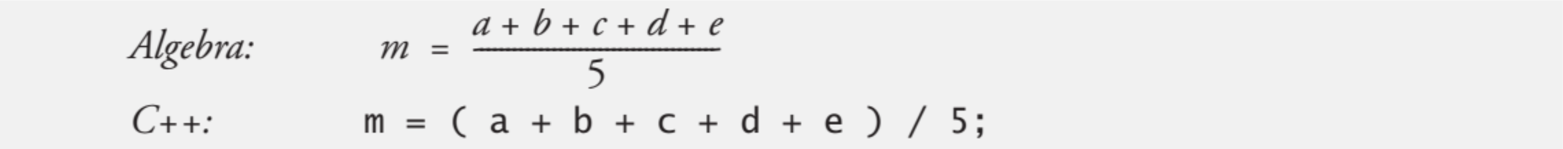


Algebra: $z = pr\%q + w/x - y$

C++:
```
z = p * r % q + w / x - y;
```

Assignment and Expression

- Rules of operator precedence
  - Example

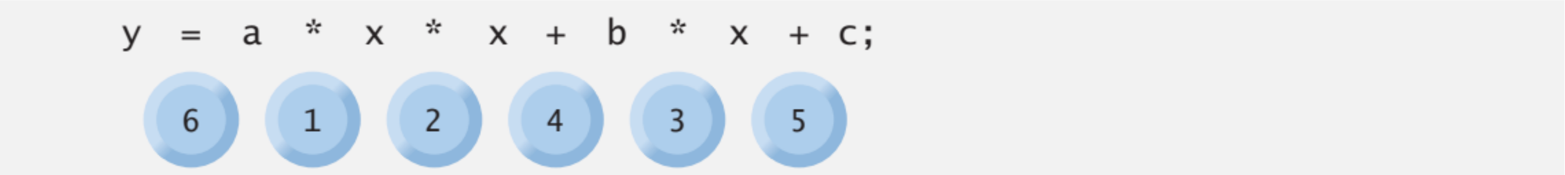


Algebra: $m = \frac{a+b+c+d+e}{5}$

C++: `m = ( a + b + c + d + e ) / 5;`

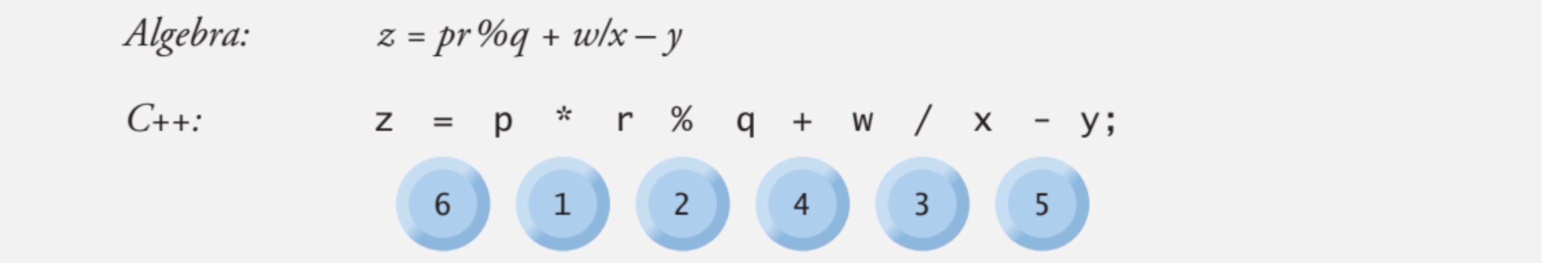
  - Incorrect solution: `m = a + b + c + d + e / 5;`

  - Example



```
y = a * x * x + b * x + c;
```

  - Example



Algebra: $z = pr \% q + w/x - y$

C++: `z = p * r % q + w / x - y;`

Assignment and Expression

- Assignment statement with same variable on both sides
  - Example:

    ```
    int total = 3;
    total = total+ 5;
    ```

  - Addition operator (+) will be executed first
    - Load the original value (3) of total to the left operand of the addition operator
    - Execute: `total + 5`

      → `3 + 5`

      → `8`
  - Assignment operator (=) will be executed next
    - Assign the value of RHS to variable total:

      `total` becomes `8`

Participation Activity 2.5.3

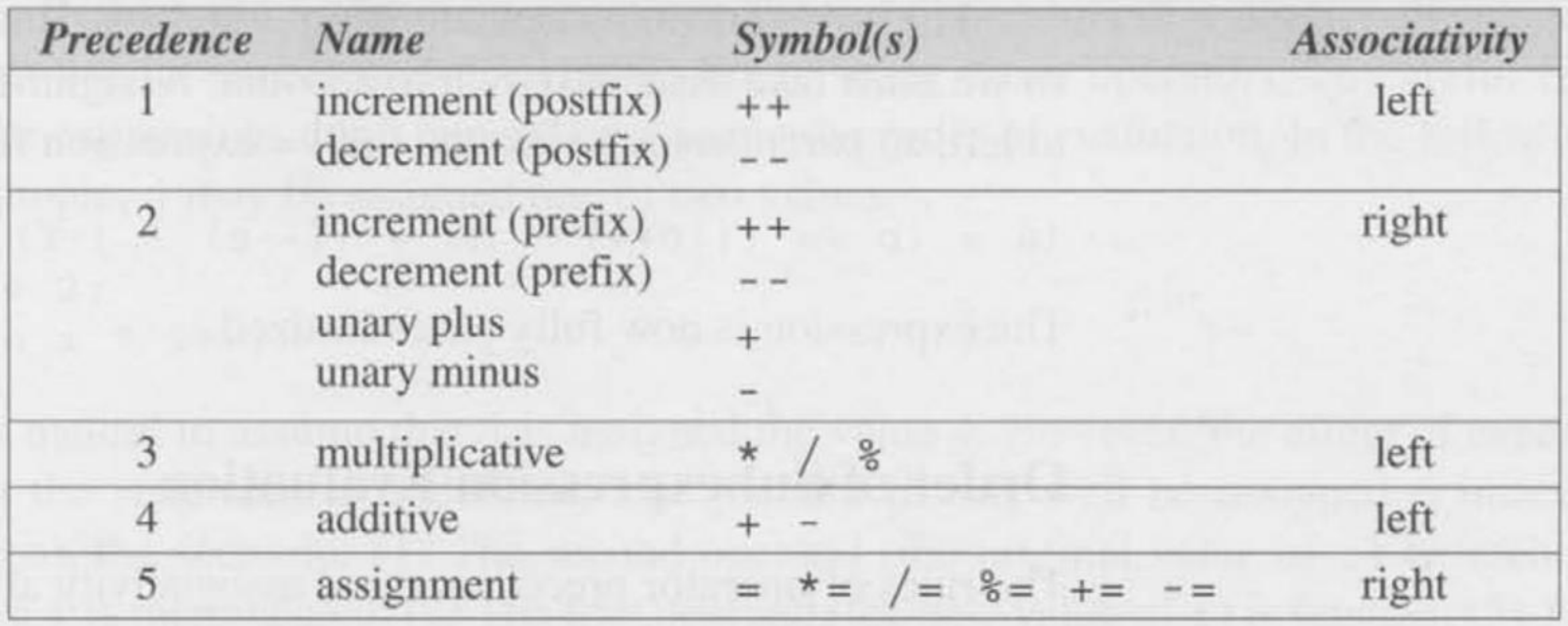
Compound Operators

- Shorthand way to update a variable
  - Example:

    ```
    i = i + 1;      is equivalent to   i += 1;
    j = j - 3;      is equivalent to   j -= 3;
    ```
  - PA
- Increment and decrement operator
  - `++`: adds 1 to its operand.
    E.g., `i++;` is equivalent to `i += 1;`
  - `--`: subtracts 1 to its operand.
    E.g., `i--;` is equivalent to `i -= 1;`

Compound Operators

- Precedence of Arithmetic Operators in C++:

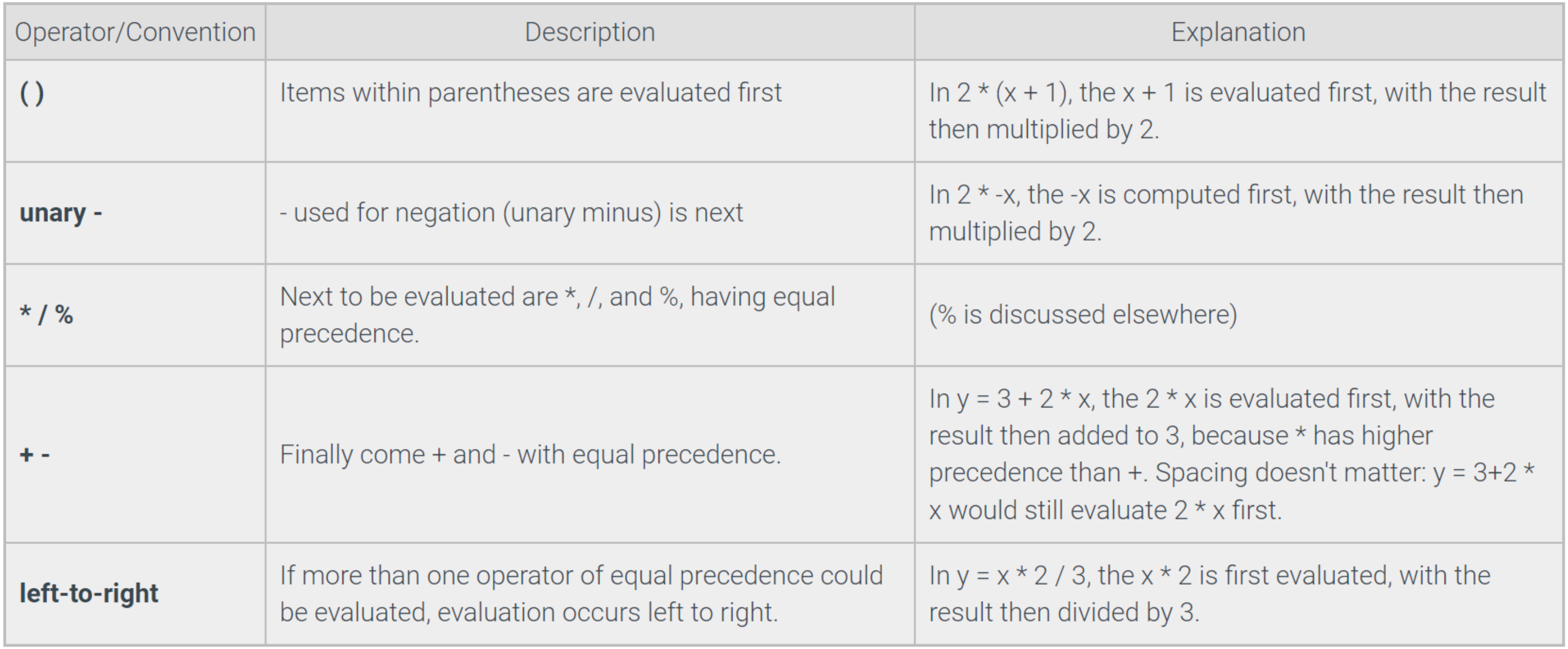


Precedence	*Name*	*Symbol(s)*	*Associativity*
1	increment (postfix) decrement (postfix)	++ --	left
2	increment (prefix) decrement (prefix) unary plus unary minus	++ -- + -	right
3	multiplicative	* / %	left
4	additive	+ -	left
5	assignment	= *= /= %= += -=	right

Math Library

Review - Assignment and Expression

- Arithmetic Operators
  - Precedence of Arithmetic Operators in C++:



Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In 2 * (x + 1), the x + 1 is evaluated first, with the result then multiplied by 2.
unary -	- used for negation (unary minus) is next	In 2 * -x, the -x is computed first, with the result then multiplied by 2.
*** / %**	Next to be evaluated are *, /, and %, having equal precedence.	(% is discussed elsewhere)
+ -	Finally come + and - with equal precedence.	In y = 3 + 2 * x, the 2 * x is evaluated first, with the result then added to 3, because * has higher precedence than +. Spacing doesn't matter: y = 3+2 * x would still evaluate 2 * x first.
left-to-right	If more than one operator of equal precedence could be evaluated, evaluation occurs left to right.	In y = x * 2 / 3, the x * 2 is first evaluated, with the result then divided by 3.

Participation Activity 1.3.1

Review - The First Program

- A simple C++ program form

```
directives

int main()
{
      statements
}
```

- Example

```
#include <iostream>
using namespace std;

int main() {
  int wage;

  wage = 20;

  cout << "Salary is ";
  cout << wage * 40 * 52;
  cout << endl;

  return 0;
}
```

Directives

- A language construct that specifies how a compiler should process its input
- In a C/C++ program, directives usually begin with a # character, which distinguishes them from other items.

#include <iostream>

- The information in **<iostream>** libraries are “included” into the program before it is compiled

<iostream>

- Contains information about C++’s console I/O library

Participation Activity 2.10.1

Math Library

- A standard math library
  - Has about 20 math operations (functions)
- Function
  - A list of statements executed by invoking the function’s name
  - Such invoking known as a function call
- How to use math library in C
  - Include `<cmath>`
  - Example

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
   double sideSquare, areaSquare = 49.0;

   sideSquare = sqrt(areaSquare);

   cout << "Square root of " << areaSquare << " is " << sideSquare << endl;

   return 0;
}
```

Auto Data Type (Since C++ 11)

Review - Variables

- Types
  - Specify what kind of data it will hold
  - Basic data types are **integers** (`short`, `int`, `long`), **real numbers** (`float` or `double`), or **characters** (`char`).
  - `int`: hold integer values, i.e., whole numbers such as 7, -11, 0, etc.
    - The largest `int` value is typically 2,147,483,647 but can be as small as 32,767
  - `float`: can store numbers with digits after the decimal point, such as 379.125
    - Slower than `int` in arithmetic operation
    - Is often an approximation of the number. E.g., `0.1` in a float variable might be 0.09999999999999987 stored in the system.
  - **Variables must be declared before they can be used**

Review - Variables

- Declaration
  - Announce the properties of variables
  - Only need to declare variable's type once. Once declared it is immutable
  - Consist of a type name and a list of variables
  - Example:

    ```
    int sum;
    int fahr, celsius;
    ```

  - Because the variables must be declared first, the simple C program form can be rewritten as

```
directives

int main()
{
    declarations
    statements
}
```

Participation Activity 2.12.1

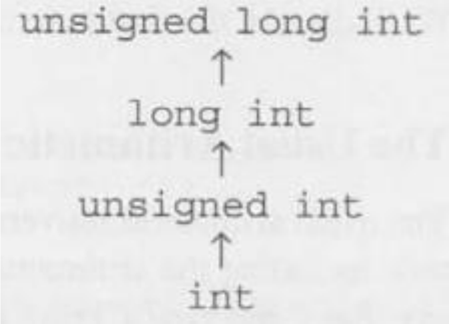
Auto Specifiers

- The compiler deduce the type from the initializer automatically
- Example
  - `auto i = 5;`
    The compiler automatically deduce the type of `i` is `int`
  - `auto j = 5.0;`
    The compiler automatically deduce the type of `j` is `double`
- Have to initialize the variable when it is declared with `auto` specifier
- Other interesting information regarding auto identifier
  - https://stackoverflow.com/questions/40781475/differences-between-c-sharp-var-and-c-auto
  - https://www.tutorialspoint.com/cplusplus/cpp_storage_classes.htm

Type Conversions

Type Conversion

- Happened when mixing different data types in operations
- Implicit conversions
  - Strategy: convert operands to the narrowest type that will safely accommodate
  - Case: the type of either operand is a floating type
    - Example: `double` + `long int` → `double`
    - Example: `float` + `long int` → `float`
  - Case: neither operand type is a floating type
    - Integral promotion: The small integral types may be converted to a larger integral type.
  - Case: both operand type is a floating type
    - Floating-point promotion: A type float can be converted to a type double



```
unsigned long int
        ↑
    long int
        ↑
  unsigned int
        ↑
       int
```

Figure 2.13.1

Type Conversion

- Explicit conversion
  - □ Use cast operator

    `static_cast<type-name>(expression)`

    type-name specifies the type to which the expression should be converted
  - □ Example:

```
int kidsInFamily1;       // Should be int, not double
int kidsInFamily2;       // (know anyone with 2.3 kids?)
int numFamilies;

double avgKidsPerFamily; // Expect fraction, so double

kidsInFamily1 = 3;
kidsInFamily2 = 4;
numFamilies = 2;

avgKidsPerFamily = static_cast<double>(kidsInFamily1 + kidsInFamily2)
                      / static_cast<double>(numFamilies);
```

Type Conversion

- Explicit conversion
  - Function-style Casting

    ```
    type-name(expression);
    ```

  - Example:

```
// initializing int variable
int num_int = 26;

// declaring double variable
double num_double;

// converting from int to double
num_double = double(num_int);
```

https://www.programiz.com/cpp-programming/type-conversion

Type Conversion

- Explicit conversion
  - C-style Type Casting

    `(data_type)expression;`
  - Example:

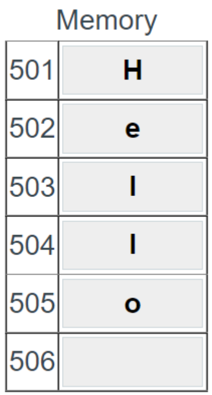
```
// initializing int variable
int num_int = 26;

// declaring double variable
double num_double;

// converting from int to double
num_double = (double)num_int;
```

https://www.programiz.com/cpp-programming/type-conversion

String

Participation Activity 2.16.1

String



Memory

501	H
502	e
503	l
504	l
505	o
506	

- A sequence of characters
  - □ String literal: surrounds a character sequence with double quotes.
  - □ Example: `"Hello"`
- A string data type isn't built into C++ like **`char`**, **`int`**, or **`double`**
  - □ Available in the standard library and can be used after adding:

    ```
    #include <string>
    ```

  - □ Declaration:

    ```
    string string_variable_name;
    ```

  - □ Example:

    ```
    string firstMonth;
    firstMonth = "January";
    cout << firstMonth << " is the first month of the year." << endl;
    ```

Participation Activity 1.3.3

Review - Basic Input/Output in C++

- standard input stream (`cin`):
  - C++ `cin` statement is the instance of the class `istream`
  - Is used to read input from the standard input device which is usually a keyboard.
  - The stream extraction operator (`>>`) is used along with the object `cin` for reading inputs.
  - Example:

    ```
    cin >> wage;
    ```

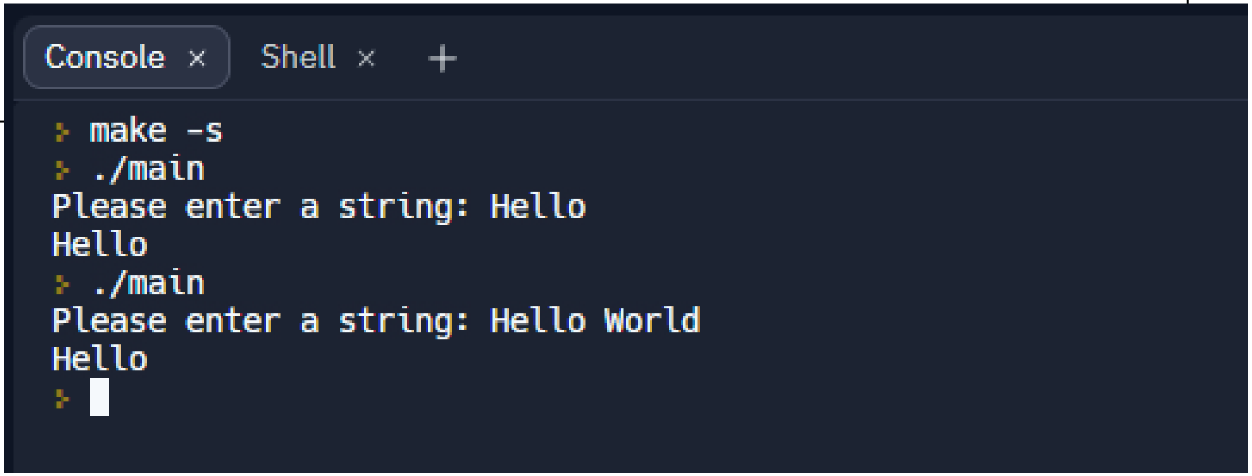
Practice zyDE 1.3.2

https://www.geeksforgeeks.org/basic-input-output-c/

Reading String Inputs

- Execute the following code with “Hello” and “Hello World” inputs:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string myString;

    cin >> myString;

    cout << myString << endl;

    return 0;
}
```



```
› make -s
› ./main
Please enter a string: Hello
Hello
› ./main
Please enter a string: Hello World
Hello
› 
```

Reading String Inputs

- Whitespace characters
  - Characters used to represent horizontal and vertical spaces
  - Includes **spaces**, **tabs**, and **newline** characters.
  - E.g., "Oh my goodness!" has two whitespace characters
- Using **`cin`** as input
  - Read characters until the first whitespace character is reached.
  - The remaining characters will be stayed in the stream buffer waiting for the next input request.
  - Skip leading whitespaces.

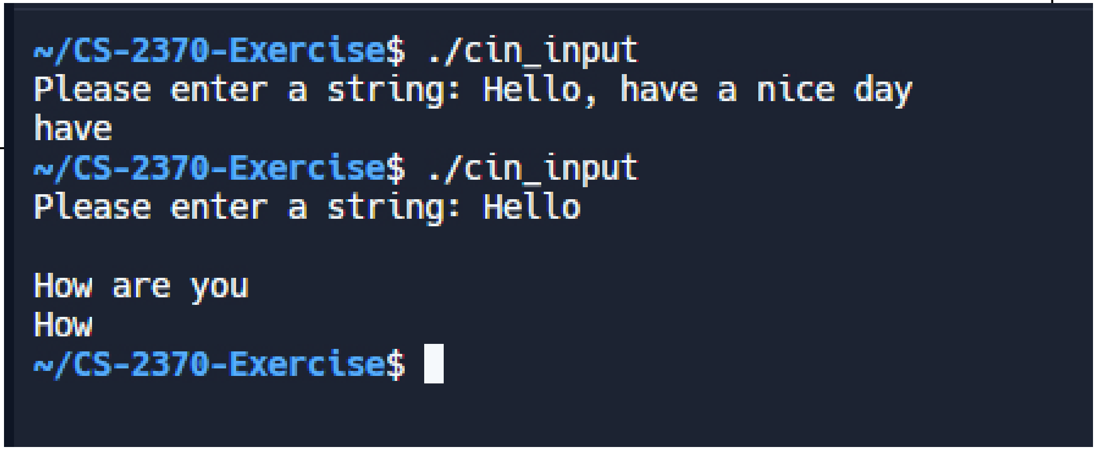
https://stackoverflow.com/questions/28109679/why-does-cin-command-leaves-a-n-in-the-buffer

Participation Activity 2.16.5

Reading String Inputs

- Example of using **cin** to read a string

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  string myString1;
  string myString2;

  cout << "Please enter a string: ";
  cin >> myString1;
  cin >> myString2;

  cout << myString2 << endl;

  return 0;
}
```

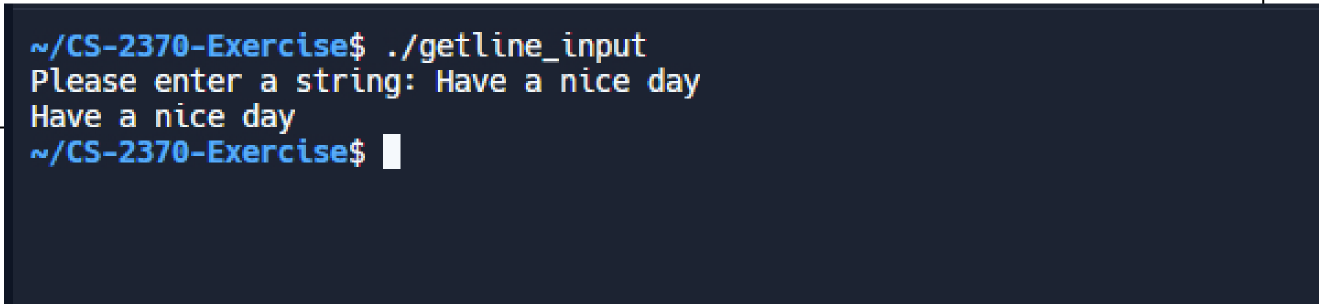


```
~/CS-2370-Exercise$ ./cin_input
Please enter a string: Hello, have a nice day
have
~/CS-2370-Exercise$ ./cin_input
Please enter a string: Hello

How are you
How
~/CS-2370-Exercise$ 
```

Reading a String with Whitespace

- Using `getline(istream, string)` function
- Example:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  string myString;

  cout << "Please enter a string: ";
  getline(cin, myString);

  cout << myString << endl;

  return 0;
}
```



```
~/CS-2370-Exercise$ ./getline_input
Please enter a string: Have a nice day
Have a nice day
~/CS-2370-Exercise$ █
```