1.1 Programming (general)

Computer program basics

Computer programs are abundant in many people's lives today, carrying out applications on smartphones, tablets, and laptops, powering businesses like Amazon and Netflix, helping cars drive and planes fly, and much more.

A computer ***program*** consists of instructions executing one at a time. Basic instruction types are:

- ***Input***: A program gets data, perhaps from a file, keyboard, touchscreen, network, etc.
- ***Process***: A program performs computations on that data, such as adding two values like x + y.
- ***Output***: A program puts that data somewhere, such as to a file, screen, network, etc.

Programs use ***variables*** to refer to data, like x, y, and z below. The name is due to a variable's value "varying" as a program assigns a variable like x with new values.

PARTICIPATION ACTIVITY 1.1.1: A basic computer program.

Animation content:

```
A generic computer program follows:
x = Get next input
y = Get next input
z = x + y
Put z to output

Input (keyboard) is as follows:
2 5

Output (screen) is as follows:
7
```


Animation captions:

1. A basic computer program's instructions get input, process, and put output. This program first assigns x with what is typed on the keyboard input, in this case 2.

2. The program's next instruction gets the next input, in this case 5.
3. The program then does some processing, in this case assigning z with x + y (so 2 + 5 yields z of 7).
4. Finally, the program puts z (7) to output, in this case to a screen.

PARTICIPATION ACTIVITY 1.1.2: A basic computer program.

Consider the example above.

1) The program has _____ instructions.

Check Show answer

2) Suppose a new instruction were inserted as follows:

...
z = x + y
Add 1 more to z (new instruction)
Put z to output

What would the last instruction then output to the screen?

Check Show answer

3) Consider the instruction: z = x + y. If x is 10 and y is 20, then z is assigned with _____.

Check Show answer

A program is like a recipe

Some people think of a program as being like a cooking recipe. A recipe consists of *instructions* that a chef executes, like adding eggs or stirring ingredients.

Likewise, a computer program consists of instructions that a computer executes, like multiplying numbers or outputting a number to a screen.

Bake chocolate chip cookies:

- Mix 1 stick of butter and 1 cup of sugar.
- Add egg and mix until combined.
- Stir in flour and chocolate.
- Bake at 350F for 8 minutes.

A first programming activity

Below is a simple tool that allows a user to rearrange some pre-written instructions (in no particular programming language). The tool illustrates how a computer executes each instruction one at a time, assigning variable m with new values throughout, and outputting ("printing") values to the screen.

PARTICIPATION ACTIVITY 1.1.3: A first programming activity.

Execute the program and observe the output. Click and drag the instructions to change the order of the instructions, and execute the program again. Not required (points are awarded just for interacting), but can you make the program output a value greater than 500? How about greater than 1000?

Run program

```
m = 5

put m

m = m * 2
put m

m = m * m
put m

m = m + 15
put m
```

m:

PARTICIPATION ACTIVITY 1.1.4: Instructions.

1) Which instruction completes the program to compute a triangle's area?
base = Get next input
height = Get next input
Assign x with base * height
_____
Put x to output

- Multiply x by 2
- Add 2 to x
- Multiply x by 1/2.

2) Which instruction completes the program to compute the average of three numbers?
x = Get next input
y = Get next input
z = Get next input
_____
Put a to output

- a = (x + y + z) / 3
- a = (x + y + z) / 2
- a = x + y + z

Computational thinking

Mathematical thinking became increasingly important throughout the industrial age, to enable people to successfully live and work. In the information age, many people believe ***computational thinking***, or creating a sequence of instructions to solve a problem, will become increasingly important for work and everyday life. A sequence of instructions that solves a problem is called an ***algorithm***.

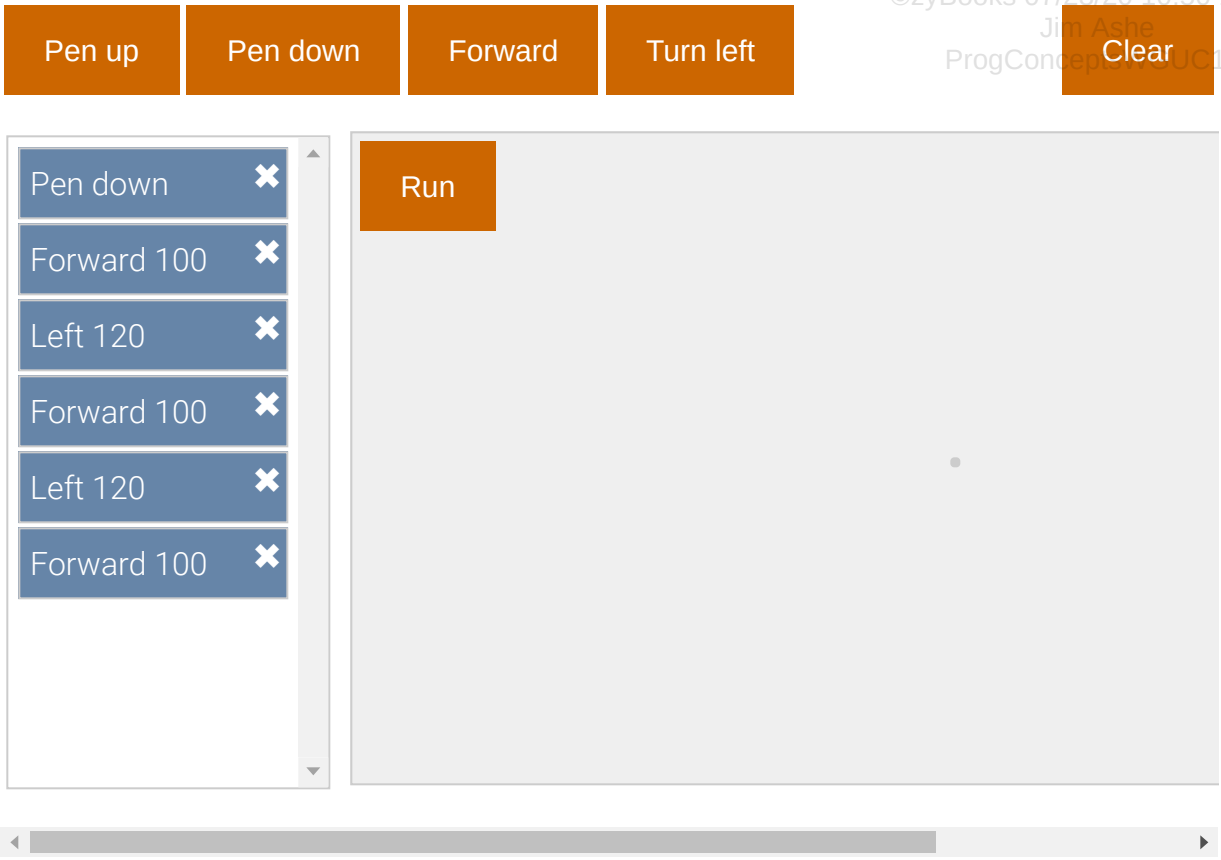
PARTICIPATION ACTIVITY 1.1.5: Computational thinking: Creating algorithms to draw shapes.

The following algorithm draws a triangle using steps like "Pen down" and "Forward 100"; press "Run" to see. Modify the algorithm to draw a square, using the following 8 steps:

1. Put the pencil down on the paper.	**5.** Turn the pencil left 90 degrees.
2. Move the pencil forward.	**6.** Move the pencil forward.

3. Turn the pencil left 90 degrees. **4.** Move the pencil forward.	**7.** Turn the pencil left 90 degrees. **8.** Move the pencil forward.

Now try to draw your initials (press "Clear" first) . Can you decompose the task of drawing your initials into a basic sequence of steps?



How was this section?

Provide feedback

1.2 Programming basics

A first program

A ***flowchart*** is a graphical language for creating computer programs. Numerous flowchart languages exist. This material defines a flowchart language, ***zyFlowchart***, intended to ease the learning of programming. A simple zyFlowchart program is shown below.

A ***program*** is a list of statements, each ***statement*** carrying out some action and executing one at a time. In zyFlowchart, each statement is in a graphical ***node***, with different shapes for different types of statements.

PARTICIPATION ACTIVITY 1.2.1: A first program.

Animation content:

```
A zyFlowchart program follows:
integer wage
integer salary

wage = 20
salary = wage * 40 * 50
Put "Salary is " to output
Put salary to output

Variables in memory is as follows:
20 wage: integer
40000 salary: integer

Output (screen) is as follows:
Salary is 40000
```

Animation captions:

1. This program has two integer "variables" for holding values, named wage and salary.
2. A program's execution begins from the start node. The first statement assigns wage with 20.
3. Execution proceeds to the next statement, which gets wage's value, and multiplies by 40 and 50 to yield 40000. The statement assigns salary with that 40000.
4. The next statement puts text "Salary is" to output. The last statement puts variable salary's value, which is 40000, to output. Execution ends when "End" is reached.

For zyFlowchart, an ***interpreter*** runs a program's statements. ***Run*** and ***execute*** are words for carrying out a program's statements.

A program also has variables, each being a name that can hold a value, like a variable x holding the value 7.

One kind of statement assigns a variable with a value; x = 7 assigns x with 7. Variable x continues to hold 7 until x is assigned a different value.

PARTICIPATION ACTIVITY 1.2.2: A first program.

Consider the program above.

1) Program execution begins at the _____ node.
   - Start
   - Begin

2) A program's statements execute _____.
   - all at once
   - sequentially

3) Variable wage was assigned with _____.
   - 20
   - 40000

4) The computation wage * 40 * 50 yielded _____.
   - 20
   - 40000

5) Each rectangle or parallelogram contains one _____.
   - statement
   - value

PARTICIPATION ACTIVITY 1.2.3: Programming terms.

Run **Variable** **Program** **Statement** **Interpreter**

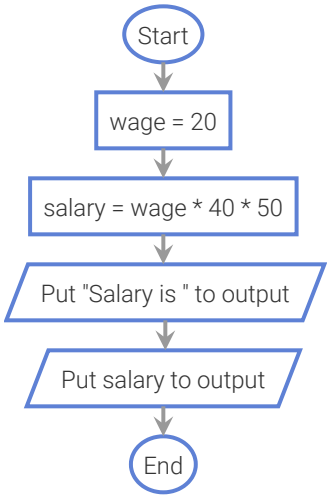
	A sequence of statements.
	A tool that runs a program's statements.

	The act of carrying out each statement's action.
	A name that can hold a value.
	A specific action in a program.

Reset

PARTICIPATION ACTIVITY 1.2.4: A first program. Full screen

Below is a web-based programming environment for zyFlowchart. Click Run to execute the program, then observe the output.



Variables

0	wage	integer
0	salary	integer

Output

ENTER EXECUTION STEP RUN Execution speed Medium

Basic input

Programs commonly get input values, then perform some processing on that input and put output values to a screen or elsewhere. Input is commonly gotten from a keyboard, a file, fields on a web form or app, etc.

In zyFlowchart, a parallelogram represents an ***input statement***, written as: `variable = Get next input`. (A parallelogram also represents an output statement, described below). The input statement assigns the indicated variable with the next program input value.

PARTICIPATION ACTIVITY 1.2.5: Getting input.

Animation content:

```
A zyFlowchart program follows:
integer wage
integer salary

wage = Get next input
salary = wage * 40 * 50
Put "Salary is " to output
Put salary to output

Variables in memory is as follows:
20 wage: integer
0 salary: integer

Input is as follows:
20

Output (screen) is as follows:
Salary is 40000
```


Animation captions:

1. A parallelogram represents an input (or output) statement.
2. The statement assigns the variable on the left (wage) with the next value in the program's input (20).
3. This program uses that gotten value in a subsequent calculation.

PARTICIPATION ACTIVITY 1.2.6: Basic input.

1) Which statement assigns variable numCars with the next input value?

   - `x = Get next input`
   - `numCars = 20`
   - `numCars = Get next input`

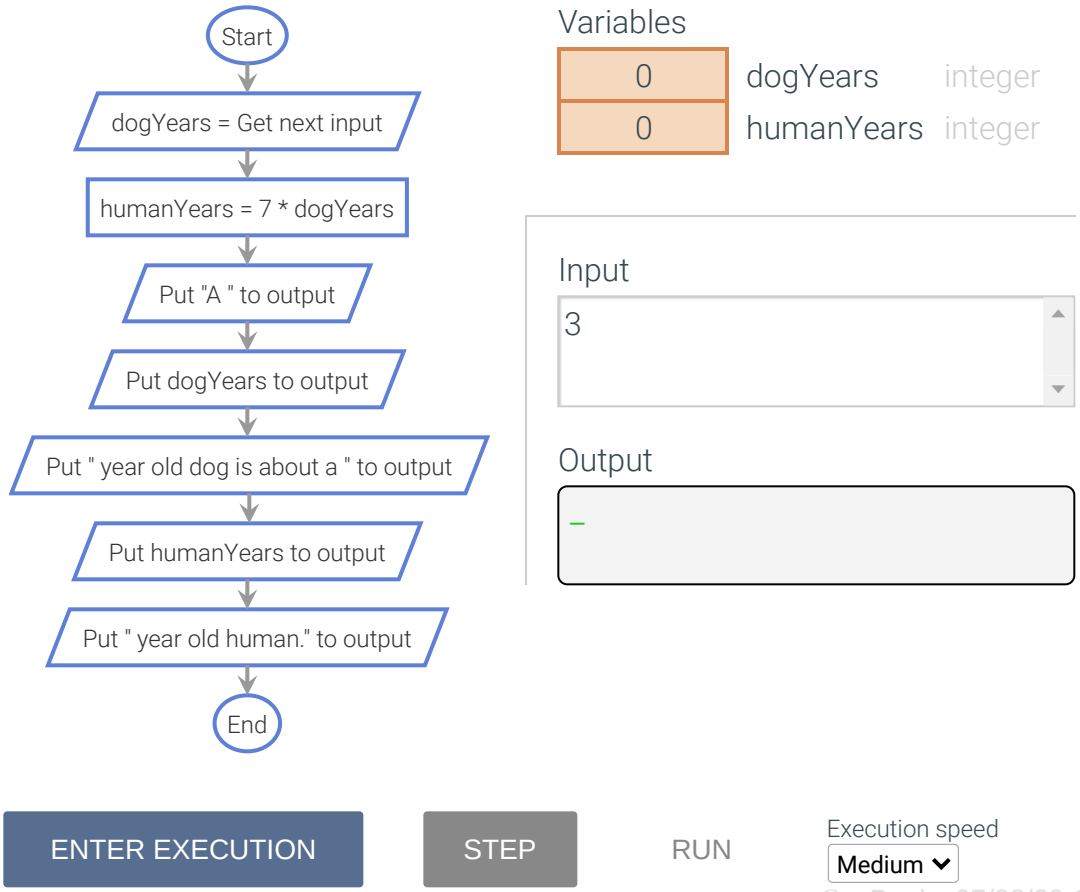
2) What shape is used for an input statement?

   - Oval
   -

- Parallelogram
- Rectangle

3) A program has statement `x = Get next input` followed by `y = Get next input`. If the program's input is 22 and 77, then:
   - x gets 22, y gets 77
   - x gets 22, y gets 22.
   - x gets 77, y gets 22.

PARTICIPATION ACTIVITY 1.2.7: Basic input. Full screen



Basic output: Text

In zyFlowchart, a parallelogram represents an ***output statement***, written as: `Put item to output`. (A parallelogram also represents an input statement, described above). The item may be a string literal or a variable. A ***string literal*** consists of text (characters) within double quotes, as in "Go #57!". A ***character*** includes any letter (a-z, A-Z), digit (0-9), or symbol (~, !, @, etc.).

A ***cursor*** indicates where the next output item will be placed in the output. The system automatically moves the cursor after the previously-output item.

A ***newline*** is a special two-character sequence \n whose appearance in an output string literal causes the cursor to move to the next output line. The newline exists invisibly in the output.

PARTICIPATION ACTIVITY 1.2.8: Outputting string literals.

Animation content:

```
A zyFlowchart program follows:
Put "Keep calm" to output
Put "\n" to output
Put "and carry on" to output

Output is as follows:
Keep calm \n
and carry on |
```

Animation captions:

1. This program puts "Keep calm" at the current cursor location in output, which initially is the beginning of the output device. The cursor automatically goes to the text's end.
2. A \n in a string literal is a special two-character sequence that outputs a 'newline' character, which is invisible but moves the cursor to the start of the next line.
3. The next output statement starts at the cursor's current location, which is now the second line.

PARTICIPATION ACTIVITY 1.2.9: String literals.

Indicate which items in a program are string literals.

1) "Hello"

- Yes
- No

2) "Hello!@#$%"

- Yes
- No

3) "Oh my"

- Yes
- No

4) " "

- Yes
- No

5) Have a nice day

- Yes
- No

6) "56"

- Yes
- No

PARTICIPATION ACTIVITY 1.2.10: Output statements.

Indicate the output of each output statement.

1) Put "Hello friend!" to output

- "Hello friend!"
- Hello friend!
- (Error)

2) Put How are you? to output

- How are you?
- (Error)

3) Put "a\nb\nc" to output

- anbnc
- a
  b
  c
- (Error)

4) Put "To be" to output
Put "or not to be" to output

- To be
  or not to be
- To be or not to be
- To beor not to be

Outputting a variable's value

Outputting a variable's value is achieved via: `Put x to output`. No quotes surround variable x's name.

The animation below replicates the first program example from earlier. The program combines outputting of both a string literal and a variable's value; such combination is common. Note that the programmer put a space at the end of "Salary is ", causing the variable's value to appear on the same line and separated from the text, as desired.

PARTICIPATION ACTIVITY | 1.2.11: Outputting a variable's value (DUPLICATE of the animation above entitled: A first program.).

Animation content:

```
A zyFlowchart program follows:
integer wage
integer salary

wage = 20
salary = wage * 40 * 50
Put "Salary is " to output
Put salary to output

Variables in memory is as follows:
20 wage: integer
40000 salary: integer

Output is as follows:
Salary is 40000
```


Animation captions:

1. This program has two integer "variables" for holding values, named wage and salary.
2. A program's execution begins from the start node. The first statement assigns wage with 20.

3. Execution proceeds to the next statement, which gets wage's value, and multiplies by 40 and 50 to yield 40000. The statement assigns salary with that 40000.
4. The next statement puts text "Salary is " to output. The last statement puts variable salary's value, which is 40000, to output. Execution ends when "End" is reached.

PARTICIPATION ACTIVITY 1.2.12: Basic variable output.

1) Given variable numCars = 9, which statement outputs 9?

- `Put "numCars" to output`
- `Get numCars to output`
- `Put numCars to output`

PARTICIPATION ACTIVITY 1.2.13: Basic variable output.

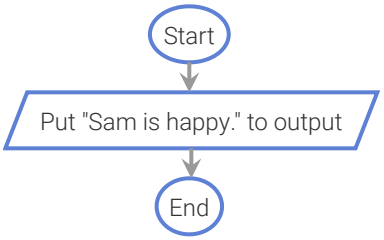
1) Type a statement that outputs the value of numUsers (a variable).

Check Show answer

CHALLENGE ACTIVITY 1.2.1: Enter the output.

Start

Type the program's output

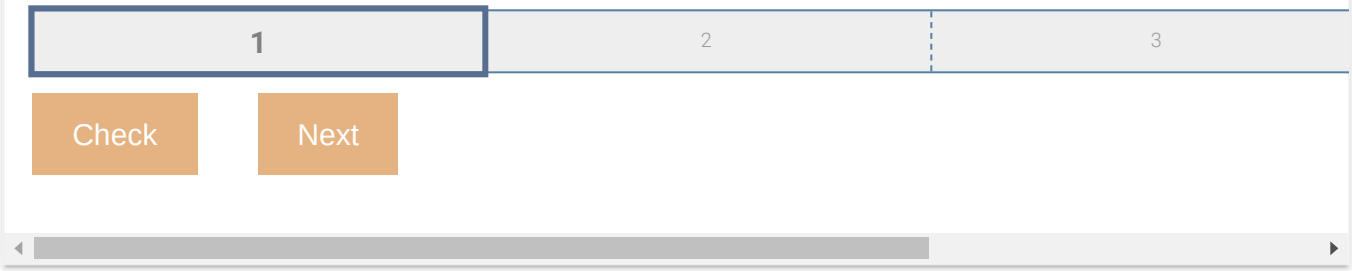


Variables

None

Output

```
Sam is happy.
```



How was this section?

Provide feedback

1.3 Comments and whitespace

Comments

A ***comment*** is text a programmer adds to a program, to be read by humans to better understand the code, but ignored by the program when executing. A comment starts with // and includes all the subsequent text on that line.

PARTICIPATION ACTIVITY 1.3.1: Comments are for humans, and ignored during execution.

Animation content:

```
A zyFlowchart program follows:
// Output header text
Put "Chapter 1" to output
// Output space below header
// Two \n's yield a blank line
Put "\n\n" to output

Output is as follows:
Chapter 1
| (next text is here)
```


Animation captions:

1. Comments begin with //. Programmers use comments to describe intent to another human reading the program.

2. During execution, the comments are entirely ignored.

PARTICIPATION ACTIVITY 1.3.2: Comments.

1) // Below computes temperature.
   - Valid comment
   - Invalid comment

2) // *** Careful!!! Tricky calculation below ***
   - Valid comment
   - Invalid comment

3) // Determine total //
   - Valid comment
   - Invalid comment

4) / Get time from user
   - Valid comment
   - Invalid comment

5) // Initialize sum with 0 before adding
   // the input values with one another
   sum = 0
   - Valid comments
   - Invalid comments

Whitespace

Whitespace refers to blank spaces (space and tab characters) between items within a statement, and to newlines. Whitespace helps improve readability for humans, but for execution purposes is mostly ignored. The following statements are equivalent, but the last one is most readable.

Figure 1.3.1: Much whitespace is ignored, so these statements perform the same computation, but the last statement is most readable.

```
w=x+y+z
```

```
w      =      x      +      y      +      z
```

```
w = x + y + z
```


A common practice is to use exactly one space before and after each operator, as in the last statement above.

PARTICIPATION ACTIVITY 1.3.3: Whitespace.

Assuming y is 3 and z is 5, Indicate with what value x is assigned. If appropriate, type: Error

1) x = y + z

Check Show answer

2) x =y+z

Check Show answer

3) x =
y + z

Check Show answer

4) x = y + z *(note the leading whitespace)*

Check Show answer

5) x = y
= + z

Check Show answer

6) num Cars = 2 5

Check Show answer

How was this section?

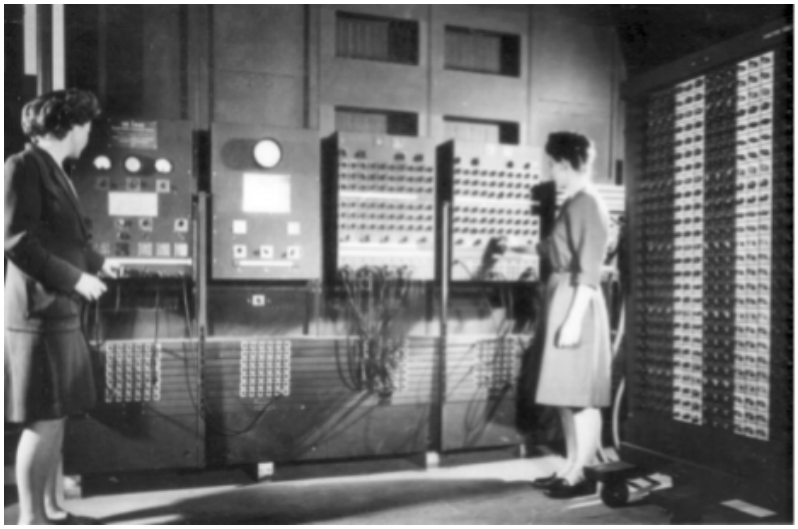
Provide feedback

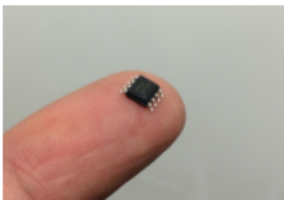
1.4 Brief history

Computers originated from telephone switches in the early 1900's. Engineers realized switches could be used for performing calculations, with a switch's on position meaning 1 and the off position 0. The first computers, built in the 1940s, had thousands of switches and occupied entire rooms. Early computers were largely used for military purposes, like codebreaking in World War II, and calculating the paths of bombs.

PARTICIPATION ACTIVITY 1.4.1: A switch is either on (1) or off (0) (click the switch).

Figure 1.4.1: 1940's computers, with thousands of baseball-sized switches, occupied entire rooms. Shrinking switches led to single-chip computers. Today, that room-sized computer fits on a chip the size of a pinhead.





Source: ENIAC computer (U. S. Army Photo / Public domain), Intel 4004 (LucaDetomi at it.wikipedia (Transfered from it.wikipedia) / GFDL or CC-BY-SA-3.0 via Wikimedia Commons), IC the size of a pinhead (zyBooks)

Engineers have reduced switch sizes by half about every 2 years, a trend known as ***Moore's Law***. Such halving is remarkable: Try repeatedly folding a sheet of paper in half and note how quickly the size shrinks (more than 7 paper folds isn't even possible). By the 1970's, an entire computer could fit on one coin-sized device known as a ***computer chip***. Today, that 1940's room-sized computer could fit on a chip smaller than a pinhead.

PARTICIPATION ACTIVITY 1.4.2: Computing history.

Some items below weren't discussed above but can be deduced.

1920's **Today** **1960's** **Moore's Law** **1940's** **1970's**

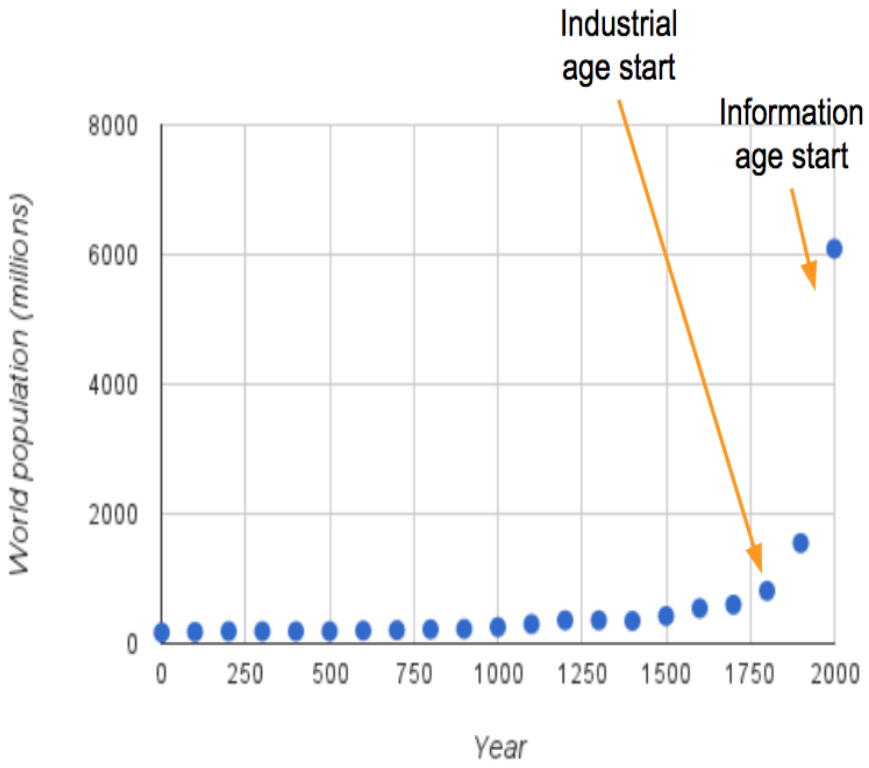
	Engineers configured telephone switches to do simple calculations like adding two numbers.

	First computers; occupied entire rooms.
	Engineers could put multiple switches on a single chip. U.S. moon rocket program brought dramatically improved chips and decreased prices.
	Earlier room-sized computers could fit on one coin-sized computer chip.
	Earlier room-sized computers could fit on a pinhead-sized chip
	The trend of switch sizes halving about every two years.

Reset

Computers are a recent phenomenon and yet are rapidly transforming human civilization. Civilization's earlier ***agricultural age*** lasted many thousands of years. The ***industrial age*** starting in the late 1700's transformed civilization towards manufacturing goods, leading to mass migration to cities, creation of strong nations, world wars, doubling of lifespans and thus dramatic world population growth (see figure below), and more. The ***information age*** just began in the 1990's, with human activity shifting from traditional industry to creating/managing/using computerized information. In just those few years, how people work, learn, communicate, or entertain themselves has already changed dramatically, but the ultimate impact on civilization likely remains to be seen.

Figure 1.4.2: The industrial age's societal transformations include a population explosion. The information age has just begun transforming society again; its ultimate impact remains to be seen.



Population data source: http://www.census.gov

PARTICIPATION ACTIVITY 1.4.3: The information age.

1) For many thousands of years, most humans lived as farmers.
   - True
   - False

2) The industrial age has led to steep population growth.
   - True
   - False

3) The information age has covered

about 100 years.

- ○ True
- ○ False

4) Society's transformation due to computers is nearly complete.

- ○ True
- ○ False

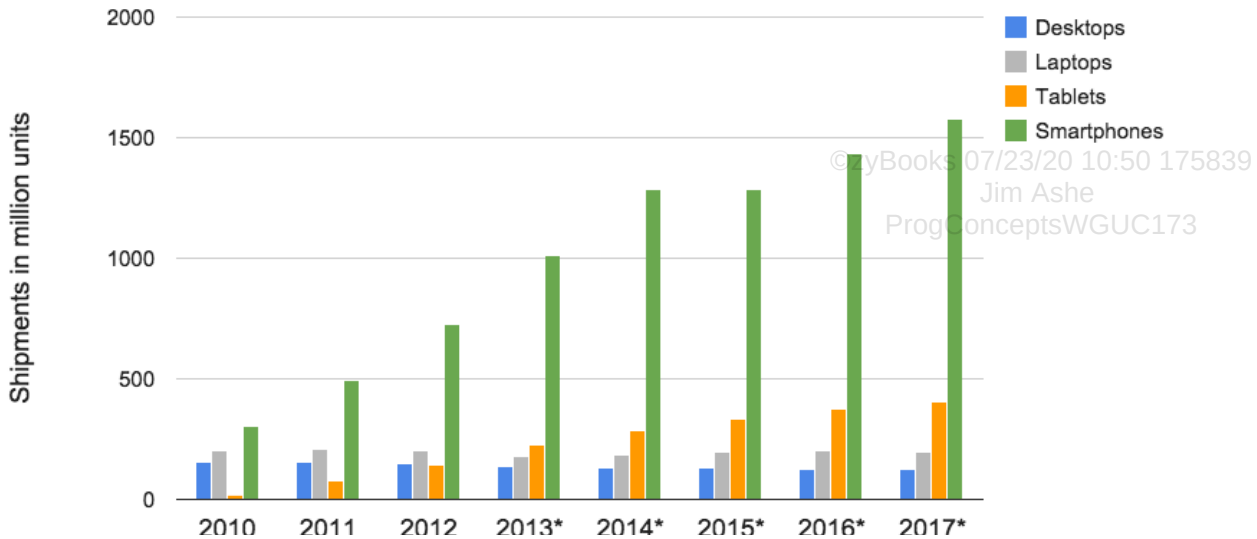
How was this section?

Provide feedback

1.5 Computers all around us

Computers in the 1940s and 1950s were large expensive machines used for government and business purposes. Initially just a few existed worldwide, costing millions of dollars (in today's dollar), weighing over a ton, and occupying entire rooms. Computer chips were invented in the 1950's, and regularly improve to have more switches and lower costs. More businesses began purchasing computers in the 1960s and 1970s. In the 1980s, people began purchasing desktop and laptop computers for home use. In the 2000's, computers began appearing in the form of electronic tablets, book readers, and smartphones too.

Figure 1.5.1: Global shipments of tablets, laptops, desktops, and smartphones.



*Forecasted values

Source: http://www.statista.com/

Figure 1.5.2: Examples of commonly-used computers: Desktops, laptops, tablets, and smartphones.

Source: zyBooks

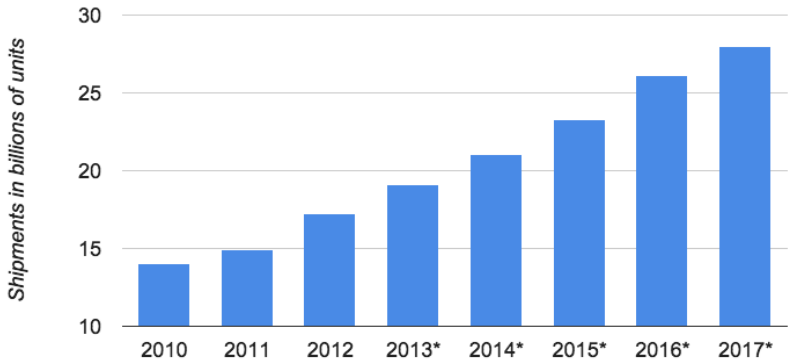
PARTICIPATION ACTIVITY 1.5.1: Computer sales.

1) The number of desktop computers continue to rise each year.
   - True
   - False

2) Which type of computer is more popular?
   - Laptops
   - Smartphones

An ***embedded computer*** is a computer inside another electrical device, like inside a TV, car, printer, thermostat, satellite, etc. Thus, beyond business and personal computing devices like PCs, tablets, and smartphones, computers exist invisibly in nearly anything electrical today too.

Figure 1.5.3: Global shipment of microcontrollers (special chips intended for embedded computers).



*Forecasted values

Source: http://www.statista.com/

Figure 1.5.4: Embedded systems: Smart pills, automobiles, and more.

Source: SmartPill™ (Given Imaging), Bullet train (Mstyslav Chernov/Unframe/www.unframe.com (Own work) / CC-BY-

SA-3.0 via Wikimedia Commons), ATMs (Rodrigo César (Own work) / CC-BY-SA-3.0 via Wikimedia Commons), Digital projector (Dave Pape (Own work) / Public domain via Wikimedia Commons), MRI (Mri magnet (Own work) / CC-BY-SA-3.0 via Wikimedia Commons), Tractor (Lifetec18 (Own work) / GFDL or CC-BY-SA-3.0 via Wikimedia Commons), Wind turbine (Kwerdenke / GFDL or CC-BY-SA-3.0 via Wikimedia Commons), all others by zyBooks

PARTICIPATION ACTIVITY 1.5.2: Embedded computers.

1) An embedded computer is a computer inside a personal computer such as a desktop or laptop.
   - True
   - False

2) Modern cars contain computers.
   - True
   - False

3) The number of embedded computers are greater than the number of desktops and laptop computers.
   - True
   - False

How was this section?

Provide feedback

1.6 Representing information as bits

Computers are built from connected switches that, like light switches, are either on or off. On is called 1, and off is 0. A single 0 or 1 is called a ***bit***. 1011 is four bits. Eight bits, like 11000101, are called a ***byte***.

Humans represent information using characters and numbers like Z or 42. Computers need a way to represent characters and numbers using 0's and 1's.

PARTICIPATION ACTIVITY 1.6.1: Bits.

1) A 0 or 1 is called a ___.

Check Show answer

2) Eight bits are called a ____.

Check Show answer

3) 101100 has ___ bits.

Check Show answer

A ***character*** is a letter (a, b, ..., z, A, B, ..., Z), symbol (!, @, #, ...), or single-digit number (0, 1, ..., 9). Basically, each item on a keyboard is a character (though more characters exist). Each character can be given a unique bit code. ***ASCII*** is a popular code for characters. Ex: Using ASCII, the letter Z would be stored in a computer as 1011010 (shown below as 101 1010; the space is just for readability in this material). Each bit code is sometimes written as an equivalent decimal number, discussed further below.

Table 1.6.1: ASCII bit codes for common characters (ignore Dec for now).

Bit code	Dec	Char	Bit code	Dec	Char	Bit code	Dec	Char
010 0000	32	space	100 0000	64	@	110 0000	96	`
010 0001	33	!	100 0001	65	A	110 0001	97	a
010 0010	34	"	100 0010	66	B	110 0010	98	b
010 0011	35	#	100 0011	67	C	110 0011	99	c
010 0100	36	$	100 0100	68	D	110 0100	100	d

010 0100	36	$	100 0100	68	D	110 0100	100	d
010 0101	37	%	100 0101	69	E	110 0101	101	e
010 0110	38	&	100 0110	70	F	110 0110	102	f
010 0111	39	'	100 0111	71	G	110 0111	103	g
010 1000	40	(	100 1000	72	H	110 1000	104	h
010 1001	41	)	100 1001	73	I	110 1001	105	i
010 1010	42	*	100 1010	74	J	110 1010	106	j
010 1011	43	+	100 1011	75	K	110 1011	107	k
010 1100	44	,	100 1100	76	L	110 1100	108	l
010 1101	45	-	100 1101	77	M	110 1101	109	m
010 1110	46	.	100 1110	78	N	110 1110	110	n
010 1111	47	/	100 1111	79	O	110 1111	111	o
011 0000	48	0	101 0000	80	P	111 0000	112	p
011 0001	49	1	101 0001	81	Q	111 0001	113	q
011 0010	50	2	101 0010	82	R	111 0010	114	r
011 0011	51	3	101 0011	83	S	111 0011	115	s
011 0100	52	4	101 0100	84	T	111 0100	116	t
011 0101	53	5	101 0101	85	U	111 0101	117	u
011 0110	54	6	101 0110	86	V	111 0110	118	v
011 0111	55	7	101 0111	87	W	111 0111	119	w
011 1000	56	8	101 1000	88	X	111 1000	120	x
011 1001	57	9	101 1001	89	Y	111 1001	121	y
011 1010	58	:	101 1010	90	Z	111 1010	122	z
011 1011	59	;	101 1011	91	[	111 1011	123	{
011 1100	60	<	101 1100	92	\	111 1100	124	|
011 1101	61	=	101 1101	93	]	111 1101	125	}

011 1110	62	>
011 1111	63	?

101 1110	94	^
101 1111	95	_

111 1110	126	~

PARTICIPATION ACTIVITY 1.6.2: ASCII bit codes (and decimal number equivalents).

Type a character: A

ASCII bit code: **1000001**

ASCII number: **65**

ASCII stands for American Standard Code for Information Interchange, and was developed in 1963. ASCII uses 7 bits per code, and has codes for 128 characters. ***Unicode*** is another character encoding standard, published in 1991, whose codes can have more bits than ASCII and thus can represent over 100,000 items, such as symbols and non-English characters.

PARTICIPATION ACTIVITY 1.6.3: ASCII.

1) What is the 7-bit code for a lower-case 'a'?

Check Show answer

2) What is the 7-bit code for a blank space?

Check Show answer

3) What two-letter word does this sequence of bits represents in ASCII? Pay attention to upper/lower case. Use the above ASCII table.
1001000 1101001

Check Show answer

4) Suppose an email message has 500 characters. How many bits would a computer use to store that email, using ASCII code?

bits

Check Show answer

Computers do many calculations, so must represent numbers. Because early humans represented values using ten fingers, humans developed base ten numbers, known as ***decimal numbers*** ("dec" refers to ten). But computers can only represent two values (0 or 1), so use base two numbers, known as ***binary numbers*** ("bi" refers to two). Rather than each digit being a power of ten and holding 0-9, each digit is a power of two and holds 0 or 1. Experiment with the below tool to understand how 0's and 1's represent numbers.

PARTICIPATION ACTIVITY 1.6.4: Understanding binary numbers.

Set each binary digit for the unsigned binary number below to 1 or 0 and observe the decimal equivalent, which is the sum of the weights of the 1 digits. Next, try to obtain the decimal equivalent of 1, then try to obtain 2, then 3. (Hint: For 3, set the two rightmost bits to 1's, the rest 0's). Try larger numbers, like your own age. Finally, set all bits to 1's and observe the largest decimal number those eight bits can represent.

0	0	0	0	0	0	0	0	**0**
128 2^7	**64** 2^6	**32** 2^5	**16** 2^4	**8** 2^3	**4** 2^2	**2** 2^1	**1** 2^0	(decimal value)

PARTICIPATION ACTIVITY 1.6.5: Binary.

1) What is binary number 1100 as a decimal number?

Check Show answer

2) What is 7 as a 4-bit binary number?

Check Show answer

CHALLENGE ACTIVITY 1.6.1: Create a binary number.

PARTICIPATION ACTIVITY 1.6.6: Characters and numbers are encoded and stored in the computer's memory.

Animation captions:

1. Each character has a unique bit code.
2. Words are stored as a sequence of bit codes in the computer's memory.
3. Symbols and spaces are also characters, and stored in the memory as bit codes.
4. Numbers are stored in binary. 12 is 0001100 in binary.

Exploring further:

- Wikipedia: ASCII
- http://www.asciitable.com/
- Wikipedia: Binary number

How was this section?

Provide feedback

1.7 Problem solving

Programming languages vs. problem solving

A chef may write a new recipe in English, but creating a new recipe involves more than just knowing English. Similarly, creating a new program involves more than just knowing a programming language. Programming is largely about ***problem solving***: Creating a methodical solution to a given task.

The following are real-life problem-solving situations encountered by one of this material's authors.

Example 1.7.1: Solving a (non-programming) problem: Matching socks.

A person stated a dislike for matching socks after doing laundry, indicating there were three kinds of socks. A friend suggested just putting the socks in a drawer, and finding a matching pair each morning. The person said that finding a matching pair could take forever: After pulling out a first sock, then pulling out a second, placing back, and repeating until the second sock matches the first, could go on for many times (5, 10, or more).

The friend provided a better solution approach: Pull out a first sock, then pull out a second, and repeat (without placing back) until a pair matches. In the worst case, if three kinds of socks exist, then the fourth sock will match one of the first three.

PARTICIPATION ACTIVITY 1.7.1: Matching socks solution approach.

Three sock types A, B, and C exist in a drawer.

1) If sock type A is pulled first, sock type B second, and sock type C third, the fourth sock type must match one of A, B, or C.
   - True
   - False

2) If socks are pulled one at a time and kept until a match is found, at least four pulls are necessary.
   - True
   - False

3) If socks are pulled two at a time and

put back if not matching, and the process repeated until the two pulled socks match, the maximum number of pulls is 4.

- True
- False

Example: Greeting people

PARTICIPATION ACTIVITY 1.7.2: Greeting people problem.

An organizer of a 64-person meeting wants to start by having every person individually greet every other person for 30 seconds. Indicate whether the proposed solution achieves the goal, without using excessive time. Before answering, think of a possible solution approach for this seemingly simple problem.

1) Form an inner circle of 32, and an outer circle of 32, with people matched up. Every 30 seconds, have the outer circle shift left one position.
   - Yes
   - No

2) Pair everyone randomly. Every 30 seconds, tell everyone to find someone new to greet. Do this 63 times.
   - Yes
   - No

3) Have everyone form a line. Then have everyone greet the person behind them.
   - Yes
   - No

4) Have everyone form a line. Have the first person greet the other 63 people for 30 seconds each. Then have the second person greet each

other person for 30 seconds each (skipping anyone already met). And so on.

- Yes
- No

5) Form two lines of 32 each, with attendees matched up. Every 30 seconds, have one line shift left one position (with the person on the left end wrapping to right). Once the person that started on the left is back on the left, then have each line split into two matched lines, and repeat until each line has just 1 person.

- Yes
- No

Example: Sorting name tags

Example 1.7.2: Example: Sorting name tags.

1000 name tags were printed and sorted by first name into a stack. A person wishes to instead sort the tags by last name. Two approaches to solving the problem are:

- Solution approach 1: For each tag, insert that tag into the proper location in a new last-name sorted stack.
- Solution approach 2: For each tag, place the tag into one of 26 sub-stacks, one for last names starting with A, one for B, etc. Then, for each sub-stack's tags (like the A stack), insert that tag into the proper location of a last-name sorted stack for that letter. Finally combine the stacks in order (A's stack on top, then B's stack, etc.)

Solution approach 1 will be very hard; finding the correct insertion location in the new sorted stack will take time once that stack has about 100 or more items. Solution approach 2 is faster, because initially dividing into the 26 stacks is easy, and then each stack is relatively small so easier to do the insertions.

In fact, sorting is a common problem in programming, and the above sorting approach is similar to a well-known sorting approach called radix sort.

PARTICIPATION ACTIVITY 1.7.3: Sorting name tags.

1000 name tags are to be sorted by last name by first placing tags into 26 unsorted sub-stacks (for A's, B's, etc.), then sorting each sub-stack.

1) If last names are equally distributed among the alphabet, what is the largest number of name tags in any one sub-stack?
   - ○ 1
   - ○ 39
   - ○ 1000

2) Suppose the time to place an item into one of the 26 sub-stacks is 1 second. How many seconds are required to place all 1000 name tags onto a sub-stack?
   - ○ 26 sec
   - ○ 1000 sec
   - ○ 26000 sec

3) When sorting each sub-stack, suppose the time to insert a name tag into the appropriate location of a sorted N-item sub-stack is N * 0.1 sec. If the largest sub-stack is 50 tags, what is the longest time to insert a tag?
   - ○ 5 sec
   - ○ 50 sec

4) Suppose the time to insert a name tag into an N-item stack is N * 0.1 sec. How many seconds are required to insert a name tag into the appropriate location of a 500 item stack?
   - ○ 5 sec
   - ○ 50 sec

©zyBooks 07/23/20 10:50 175839
Jim Ashe
ProgConceptsWGUC173

A programmer usually should carefully create a solution approach *before* writing a program. Like English being used to describe a recipe, the programming language is just a description of a solution approach to a problem; creating a good solution should be done first.

How was this section?

Provide feedback

1.8 Why programming

Computing careers

While careers in law, medicine, and engineering have existed for hundreds of years, computers are relatively new so careers in computing are new too. Today, computing jobs are often ranked among the best jobs, in terms of opportunity, salary, work-life balance, job security, job satisfaction, work conditions, etc. Nearly all computing jobs require some training in programming; some jobs then focus on programming, while others instead focus on related aspects.

In a 2017 ranking (below), 2 of the top 15 jobs were computing jobs. In another ranking, 3 of the top 20 were computing jobs. Note: Rankings from different sources vary greatly; some have more engineers, human resources managers, data scientists, marketing, etc. Also, the specific ordering in a ranking is not usually substantial (like rank #2 vs. #5), and rankings change every year. However, note that most rankings consistently have several computing jobs in the top tier.

Table 1.8.1: Best jobs of 2017, per U.S. News and World Report.

Ranking	Occupation	Description
1-7	Dentist, nurse practitioner, physician's assistant, statistician, orthodontist, nurse anesthetist, pediatrician	
8	Computer systems analyst	Helps companies evaluate and improve computer systems: PCs, networking, servers, laptops, tablets, etc.
9-12	Obstetrician/gynecologist, oral surgeon, optometrist, occupational	

9-12	surgeon, optometrist, occupational therapy assistant,	
13	Software developer	Designs computer programs, combining creativity and technical know-how, often working in teams.
14,15	Surgeon, nurse midwife	

Source: U.S. News and World Report (includes links to expanded descriptions).

PARTICIPATION ACTIVITY 1.8.1: Computing jobs are often ranked among the best jobs.

1) What one factor was used to rank the best jobs?
   - Salary
   - Job security
   - Multiple factors were considered

2) A computer systems analyst mainly does what?
   - Writes software
   - Evaluates and improves computer systems
   - Repairs computers

3) Software developers spend nearly all their time alone at a computer.
   - True
   - False

4) Interestingly, the above list is dominated by jobs in what two general areas?
   - Computing, and health care
   - Computing, and manufacturing

Types of computing jobs

Table 1.8.2: Computing jobs.

A wide variety of computing jobs exist.

Occupation	Job Summary	Entry-level education	2016 median pay
Computer and Information Research Scientists	Computer and information research scientists invent and design new approaches to computing technology and find innovative uses for existing technology. They study and solve complex problems in computing for business, medicine, science, and other fields.	Doctoral or professional degree	$111,840
Computer Network Architects	Computer network architects design and build data communication networks, including local area networks (LANs), wide area networks (WANs), and intranets. These networks range from a small	Bachelor's degree	$101,210

	connection between two offices to a multinational series of globally distributed communications systems.		
Computer Programmers	Computer programmers write code to create software programs. They turn the program designs created by software developers and engineers into instructions that a computer can follow.	Bachelor's degree	$79,840
Computer Support Specialists	Computer support specialists provide help and advice to people and organizations using computer software or equipment. Some, called computer network support specialists, support information technology (IT) employees within their organization. Others, called	Varies: High-school degree and higher	$52,160

	computer user support specialists, assist non-IT users who are having computer problems.		
Computer Systems Analysts	Computer systems analysts study an organization's current computer systems and procedures and design information systems solutions to help the organization operate more efficiently and effectively. They bring business and information technology (IT) together by understanding the needs and limitations of both.	Bachelor's degree	$87,220
Database Administrators	Database administrators (DBAs) use specialized software to store and organize data, such as financial information and customer shipping records. They	Bachelor's degree	$84,950

	make sure that data are available to users and are secure from unauthorized access.		
Information Security Analysts	Information security analysts plan and carry out security measures to protect an organization's computer networks and systems. Their responsibilities are continually expanding as the number of cyberattacks increase.	Bachelor's degree	$92,600
Network and Computer Systems Administrators	Computer networks are critical parts of almost every organization. Network and computer systems administrators are responsible for the day-to-day operation of these networks.	Bachelor's degree	$79,700
Software Developers	Software developers are the creative minds behind computer programs. Some develop the	Bachelor's degree	$102,280

	develop the applications that allow people to do specific tasks on a computer or other device. Others develop the underlying systems that run the devices or control networks.		
Web Developers	Web developers design and create websites. They are responsible for the look of the site. They are also responsible for the site's technical aspects, such as performance and capacity, which are measures of a website's speed and how much traffic the site can handle. They also may create content for the site.	Associate's degree	$66,130

Source: bls.gov (includes links to detailed descriptions and outlooks for each occupation).

PARTICIPATION ACTIVITY 1.8.2: Computing jobs.

Refer to the above BLS table of computing jobs.

Web developers | Computer systems analysts | Software developers

Information security analysts | Computer programmers

Computer support specialists

	Likely requires both a strong knowledge of computer technology, and excellent interpersonal skills due to dealing with non-technical users.
	Create, design, and program software.
	Help write programs created by software developers.
	Help organizations use computing technology to operate effectively. Requires strong combination of business and computing technology knowledge.
	Focus on protecting an organization's computers and data. Increasingly important as "hackers" continue to steal huge amounts of data, as widely-publicized in recent years.
	Build websites, which may involve the look/feel, the content, the performance of the site, and more.

©zyBooks 07/23/20 10:50 175839
Jim Ashe
ProgConceptsWGUC173

Reset

For many non-computing jobs (dentist, attorney, nurse, business, etc.), computer usage is high, and thus knowledge of computing technology can yield strong advantages even for people not in a computing career.

Programming and non-computing jobs

Many people in non-computing jobs find that knowing some programming can benefit their careers. Some examples:

- *Kelly* majored in chemistry, and now works as a scientist in a pharmaceutical company. Kelly helps analyze clinical trials. Her company uses commercial statistical software, but she found that writing small custom programs yielded even better analyses. Her co-workers now come to her for help. She is glad she took a required programming class in college, though at the time she wasn't as happy about it.
- *Paul* majored in civil engineering, and now authors technical content for a large company. Paul noticed that several authoring tasks done in Google Docs by the in-house 25-person authoring team could be automated. Building on the programming he learned in a required college course, Paul spent several hours online learning about Google Docs "add on" programming, and wrote two small add-ons. His add-on programs have become part of the standard authoring process for the entire team, who frequently thanks Paul for saving them time and relieving them of tedious tasks.
- *Ethan* majored in business, and got a job in sales operations of a Silicon Valley startup company. Building on the C++ programming he learned from a college course, he started tinkering with writing database query programs using "SQL", and discovered he had a knack for it. His job duties have expanded to include running database reports, and he has automated dozens of reports via programming, helping people throughout the company be more productive.
- *Eva* (pictured above) majored in environmental science. She voluntarily took a programming course in college believing the knowledge/skills could be important to her. She took a job at a startup company doing various marketing tasks. She began to manage the company's website, and realized that a few small programs could make the web pages dynamic and interactive. She wrote the code herself, which was reviewed and approved by the engineering team and became part of the company's live website. She plans on getting a graduate degree in environment science and expects to programming will be useful in her research.

PARTICIPATION ACTIVITY 1.8.3: Programming in non-computing jobs.

Consider the examples above.

1) Kelly voluntarily took a programming

course in college.

- True
- False

2) Ethan learned SQL programming in a college course and now applies SQL programming in his job.

- True
- False

3) Eva voluntarily took a programming class in college.

- True
- False

Precision, logic, and computational thinking

Many people find that programming encourages precise, logical thought that can lead to better writing and speaking, clearer processes, and more. The thought processes needed to build correct, precise, logical programs is sometimes called ***computational thinking*** and has benefits beyond programming.

PARTICIPATION ACTIVITY 1.8.4: Learning programming tends to aid in precise, logical thought, aspects of computational thinking.

Animation captions:

1. Common English usage may be vague. Are workers, painters, and contractors the same people or different? What exactly is white and brown?
2. Programs use one word per item; no synonyms, no pronouns. In English, using "painters" consistently, and replacing "they" with "The IDs", yields precise info.
3. Policies and other documents often aren't logical, with conflicting or missing info. How can a person 20 miles away take a taxi if they must drive? What about 100 miles?
4. Programmers use precise structures like "If-else" statements. When used in English, the result is logical, unambiguous info. Some call this "computational thinking".

New programmers often complain about how unforgiving programming is, but such attention to detail is one of the benefits of learning programming.

PARTICIPATION ACTIVITY 1.8.5: Computational thinking.

1) What's wrong with this survey question?

How many minutes did you spend?
__ Under 5
__ 6 or more

- Should say "More than 6" instead of "6 or more".
- Exactly 5 minutes is not a choice

2) An online shopping site allows setting up automated repeat purchases. A person needs to determine the rate for laundry detergent. One bottle does 64 loads. He does a load a week. His wife does a load a week. His daughter does a load every two weeks. What's the best rate?

- Every 24 weeks
- Every 32 weeks
- Every 64 weeks

You've never done anything like this

Programming is different than nearly anything most students have done before. Most new programmers initially struggle. Just as a child learning to walk will stumble and fall, a student learning to program will stumble and fall many times as well.

Programs have literally transformed the world in the past few decades. But, *correct programs are hard to create*. Programs are among the most sophisticated of human creations. Even one wrong symbol in a program with thousands of characters can cause the program to entirely fail. And programs deal with doing long sequences of tasks over time. Such features are not common in other aspects of life.

Programming is a combination of concepts and skill. The skill part is not as common in other "academic" subjects. Learning to program thus requires practice. A student cannot watch a piano teacher play and then walk away playing piano. Writing correct expressions, properly formed if-else branches, correctly working loops, etc., requires repeated attempts, and, like the new piano player, lots of mistakes along the way.

Programming also requires a lot of mental energy. No easy steps exist for how to solve a given problem by writing a program. Many students are not accustomed to having to think so hard to solve a problem, instead looking to follow standard steps or just trying to "look up the answer".

Students studying programming are about to embark on one of the most rewarding but also the most challenging of human endeavours. When stuck, students may wish to take solace that everyone struggles. Like the child learning to walk, each fall hurts, but know that each fall brings one closer to learning a powerful skill.

Even the best programmers make mistakes

Even the best programmers make mistakes. In San Diego 2012, a software bug caused 17-minutes of fireworks to launch nearly simultaneously.

Video 1.8.1: When software goes wrong...

PARTICIPATION ACTIVITY 1.8.6: Programming.

1) For most people, programming comes easy.

- ○ True
- ○ False

2) If a student has trouble converting a problem statement into a program, the teacher and/or learning content must have done a poor job.

- True
- False

How was this section?

Provide feedback

1.9 Pseudocode: Basics

Pseudocode is text that resembles a program in a real programming language but is simplified to aid human understanding. "Code" is a term for a program written in text. "Pseudo" in this context means "similar to". Commonly, each pseudocode text line corresponds to a program statement.

Compared to a real textual programming language like C, C++, Java, or Python, pseudocode may omit various details and/or use a more sentence-like notation to aid human understanding.

Compared to a graphical programming language, pseudocode's text may be easier to create (just by typing) and to share than graphics. Also, pseudocode may be more compact, thus being easier to view and understand, especially for larger programs.

PARTICIPATION ACTIVITY 1.9.1: Pseudocode compactly represents a program's statements.

Animation content:

```
The following program is shown, in graphical form on the left, and
textual form on the right.
wage = 20
salary = wage * 40 * 50
Put "Salary is " to output
Put salary to output
```


Animation captions:

1. Pseudocode describes a program in text form, with one statement per line of text. The text may be easier to create than graphics, and more compact.
2. The pseudocode is considered to execute one statement at a time.

PARTICIPATION ACTIVITY 1.9.2: Pseudocode.

1) Pseudocode refers to an informal textual description of a program.
   - True
   - False

2) Pseudocode is intended primarily for machines to read and execute the program.
   - True
   - False

3) In the pseudocode above, how many statements exist?
   - 1
   - 4

How was this section?

Provide feedback

1.10 Introduction summary

This chapter's key points included:

- A program consists of instructions (aka statements) that execute one at a time, to get input, process data, and put output.
- A program uses variables to hold data, which may change ("vary").
- A flowchart depicts a program graphically, with a node for each statement.
- A program can output a variable's value, or a string literal consisting of characters (including a newline character).
- A program can contain comments, which are for humans only and ignored when the program runs.

- Most whitespace (regular spaces or newlines) is ignored by a program, but good practice uses whitespace in a consistent way.
- The information age is quite new in human history. Changes are rapid.
- Computers surround us and computer numbers and usage continue to grow.
- Inside a computer, all data (characters, numbers, and more) is represented as bits: 0's and 1's.
- Programming is largely about problem solving, namely creating a methodical solution to a given task.
- Careers in computing are numerous, highly-rated, and growing. Non-computing jobs may benefit from programming.
- Pseudocode is an informal textual representation of a program intended for easy human understanding.

PARTICIPATION ACTIVITY 1.10.1: Introduction summary.

1) Each program statement either gets input, _____ data, or puts output.
   - counts
   - processes
   - throws away

2) A variable's value _____.
   - may change
   - does not change

3) The information age _____.
   - ended last century
   - is nearing its end
   - will likely continue for a while

4) Programming is largely about _____.
   - guessing
   - problem solving
   - drawing graphical programming nodes

5) Pseudocode represents a program _____.
   - graphically and formally
   - graphically and informally

- textually and informally

How was this section?

Provide feedback

©zyBooks 07/23/20 10:50 175839
Jim Ashe
ProgConceptsWGUC173

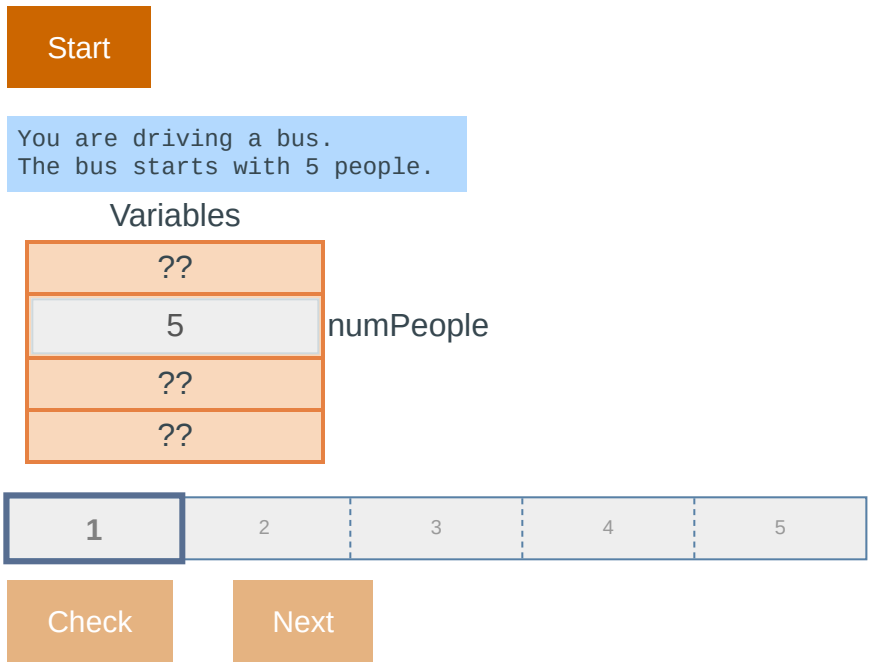
2.1 Variables and assignments (general)

Remembering a value

Here's a variation on a common schoolchild riddle.

PARTICIPATION ACTIVITY 2.1.1: People on bus.

For each step, keep track of the current number of people by typing in the numPeople box (the box is editable).



By the way, the real riddle ends with the question "What is the bus driver's name?" and the subject usually says "How should I know?". The riddler then says "I started with YOU are driving a bus."

The numPeople box above served the same purpose as a *variable* in a program, introduced below.

Variables and assignments

In a program, a ***variable*** is a named item, such as x or numPeople, used to hold a value.

An ***assignment statement*** assigns a variable with a value, such as x = 5. That statement means x is assigned with 5, and x keeps that value during subsequent statements, until x is assigned

again.

An assignment statement's left side must be a variable. The right side is an expression. Example statements are x = 5, y = a, or z = w + 2. The 5, a, and w + 2 are each an expression that evaluates to a value.

PARTICIPATION ACTIVITY 2.1.2: Variables and assignments.

Animation content:

```
A generic program follows:
x = 5
y = x
z = x + 2
x = 3

Variables in memory is as follows:
5 (greyed out) 3 x
5 y
7 z
```

Animation captions:

1. In programming, a variable is a place to hold a value. Here, variables x, y, and z are depicted graphically as boxes.
2. An assignment statement assigns the left-side variable with the right-side expression's value. x = 5 assigns x with 5.
3. y = x assigns y with x's value, which presently is 5. z = x + 2 assigns z with x's present value plus 2, so 5 + 2 or 7.
4. A subsequent x = 3 statement assigns x with 3. x's former value of 5 is overwritten and thus lost. Note that the values held in y and z are unaffected, remaining as 5 and 7.
5. In algebra, an equation means "the item on the left always equals the item on the right". So for x + y = 5 and x * y = 6, one can determine x = 2 and y = 3.
6. Assignment statements look similar to algebra equations but have VERY different meaning. The left side MUST be one variable.
7. The = isn't "equals", but is an action that PUTS a value into the variable. Assignment statements only make sense when executed in sequence.

= is not equals

In programming, = is an assignment of a left-side variable with a right-side value. =

is NOT equality as in mathematics. Thus, x = 5 is read as "x is assigned with 5", and not as "x equals 5". When one sees x = 5, one might think of a box labeled x having the value 5 put in.

PARTICIPATION ACTIVITY 2.1.3: Valid assignment statements.

Indicate which assignment statements are valid.

1) x = 1

- Valid
- Invalid

2) x = y

- Valid
- Invalid

3) x = y + 2

- Valid
- Invalid

4) x + 1 = 3

- Valid
- Invalid

5) x + y = y + x

- Valid
- Invalid

PARTICIPATION ACTIVITY 2.1.4: Variables and assignment statements.

Given variables x, y, and z.

1) x = 9
y = x + 1
What is y?

Check Show answer

2) x = 9
y = x + 1
What is x?

Check Show answer

3) x = 9
y = x + 1
x = 5
What is y?

Check Show answer

PARTICIPATION ACTIVITY 2.1.5: Trace the variable value.

Select the correct value for x, y, and z after the following statements execute.

Start

```
x = 3
y = 7
z = 0
x = 2
y = 8
z = 6
x = 4
```

x is		
2	4	3

y is		
7	8	5

z is		
5	0	6

1 2 3 4

Check Next

Assignments with variable on left and right

Because = means assignment in programming, a variable may appear on both the left and right side of a statement, as in x = x + 1. If x was originally 6, x is assigned with 6 + 1, or 7. The statement overwrites the original 6 in x.

PARTICIPATION ACTIVITY 2.1.6: A variable may appear on the left and right of an assignment statement.

Animation content:

```
A generic program follows:
x = 1
x = x * 20
x = x * 20
Put "A person with measles may cause " to output
Put x to output
Put newline to output
Put "people to be infected in two weeks." to output

Variables in memory is as follows:
1 (greyed out) 20 (greyed out) 400 x

Output is as follows:
A person with measles may cause 400
people to be infected in two weeks.
```

Animation captions:

1. A variable may appear on both sides of an assignment statement. After x = 1, then x = x * 20 assigns x with 1 * 20 or 20, overwriting x's previous value 1.
2. Another x = x * 20 assigns x with 20 * 20 or 400, which overwrites x's previous value 20.
3. Only the latest value is held in x. The previous values are shown greyed out above but in actuality are completely gone.

PARTICIPATION ACTIVITY 2.1.7: Variable on both sides.

Indicate the value of x after the statements execute.

1) x = 5
   x = x + 7

Check Show answer

2) x = 2
y = 3

x = x * y
x = x * y

Check Show answer

3) y = 30
x = y + 2
x = x + 1

Check Show answer

How was this section?

Provide feedback

2.2 Variables (integer)

Variable declarations

A ***variable declaration*** declares a new variable, specifying the variable's name and type. A variable of type ***integer*** can hold whole number values, like 1, 999, 0, or -25 (but not 3.5 or 0.001). In zyFlowcharts, an integer variable's initial value is 0. The example program below declares a variable named userAge that can hold an integer value.

When a statement that assigns a variable with a value executes, the program puts the value into the variable. The variable will hold that value until the variable is assigned with another value.

Because computer memory is limited, an integer cannot hold arbitrarily large values. In zyFlowchart, as in many programming languages, the range is about -2 billion to +2 billion.

PARTICIPATION ACTIVITY 2.2.1: Variable declaration and assignment.

Animation content:

```
A zyFlowchart program follows:
integer userAge

userAge = Get next input
Put userAge to output
Put " is a great age." to output

Variables in memory is as follows:
23 userAge: integer

Input is as follows:
23

Output is as follows:
23 is a great age.
```


Animation captions:

1. The programmer declares a variable named userAge with the type integer. The variable is automatically initialized with the value 0.
2. The program gets input and assigns userAge with 23.
3. The program puts userAge's value, or 23, to output.

PARTICIPATION ACTIVITY 2.2.2: Declaring integer variables.

Refer to the above animation.

1) What is userAge's type?

- integer
- number
- float

2) What was userAge's initial value?

- 0
- 23
- Unknown

3) What was userAge's final value?

- 0
- 23
- Unknown

Assignment statements

An ***assignment statement*** assigns the variable on the left-side of the = with the current value of the right-side expression. Ex: `numApples = 8` assigns numApples with the value of the right-side expression (in this case 8).

An ***expression*** may be a number like 80, a variable name like numApples, or a simple calculation like numApples + 1. Simple calculations can involve standard math operators like +, -, and *, and parentheses as in 2 * (numApples - 1).

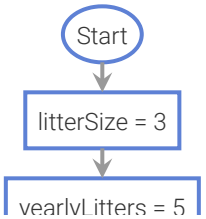
In the code below, litterSize is assigned with 3, and yearlyLitters is assigned with 5. Later, annualMice is assigned with the value of litterSize * yearlyLitters (3 * 5, or 15), which is then printed. Next, litterSize is assigned with 14, yearlyLitters is assigned with 10, and annualMice is assigned with their product (14 * 10, or 140), which is printed.

PARTICIPATION ACTIVITY 2.2.3: Assigning a variable. Full screen

Variables

0	litterSize	integer
0	yearlyLitters	integer
0	annualMice	integer

Output



ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 2.2.4: Assignment statements.

1) Write an assignment statement to assign numCars with 99.

Check Show answer

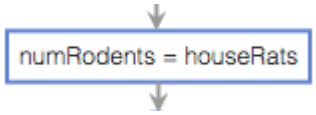
2) Assign houseSize with 2300.

Check Show answer

3) Assign numFruit with the current value of numApples.

Check Show answer

4) The current value in houseRats is 200. What is in houseRats after executing the statement below? Valid answers: 0, 199, 200, or unknown.

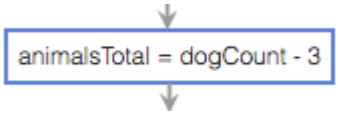


Check Show answer

5) Assign numItems with the result of ballCount - 3.

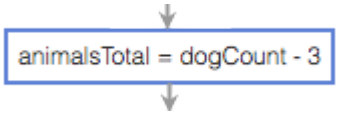
Check Show answer

6) dogCount is 5. What is in animalsTotal after executing the statement below?



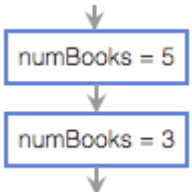
Check Show answer

7) dogCount is 5. What is in dogCount after executing the statement below?



Check Show answer

8) What is in numBooks after both statements execute?



Check Show answer

Assignment statement with same variable on both sides

Commonly, a variable appears on both the right and left side of the = operator. Ex: If numItems is 5, after `numItems = numItems + 1` executes, numItems will be 6. The statement reads the value of numItems (5), adds 1, and assigns numItems with the result of 6, which replaces the value previously held in numItems.

PARTICIPATION ACTIVITY | 2.2.5: Variable assignments overwrite a variable's previous values: People-known example.

Animation content:

```
A zyFlowchart program follows:
integer yourFriends
integer totalFriends

yourFriends = Get next input
totalFriends = yourFriends * yourFriends
Put "Your friends know " to output
Put totalFriends to output
Put " people.\n" to output
totalFriends = totalFriends * yourFriends
Put "And, they know " to output
Put totalFriends to output
Put " people.\n" to output

Variables in memory is as follows:
200 yourFriends: integer
8000000 totalFriends: integer

Input is as follows:
200

Output is as follows:
Your friends know 40000 people.
And, they know 8000000 people.
```


Animation captions:

1. The program declares two integer variables yourFriends and totalFriends.
2. The program gets input and assigns yourFriends with 200.
3. The assignment statement reads yourFriends (200), multiplies yourFriends by yourFriends, and assigns totalFriends with the product of 40000.
4. The program outputs totalFriends, which holds the total people your friends' know.

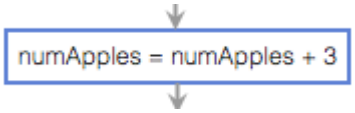
5. Assignment reads totalFriends (40000) and yourFriends (200), multiplies those values, and assigns totalFriends with the result of 8000000.
6. The program outputs totalFriends, which now holds the total people your friends' friends know.

Six degrees of separation

The above example relates to the popular idea that any two people on earth are connected by just "six degrees of separation", accounting for overlapping of known-people.

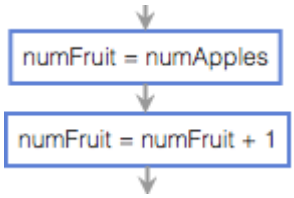
PARTICIPATION ACTIVITY 2.2.6: Assignment statements with same variable on both sides.

1) numApples is initially 5. What is numApples after:



Check Show answer

2) numApples is initially 5. What is numFruit after:

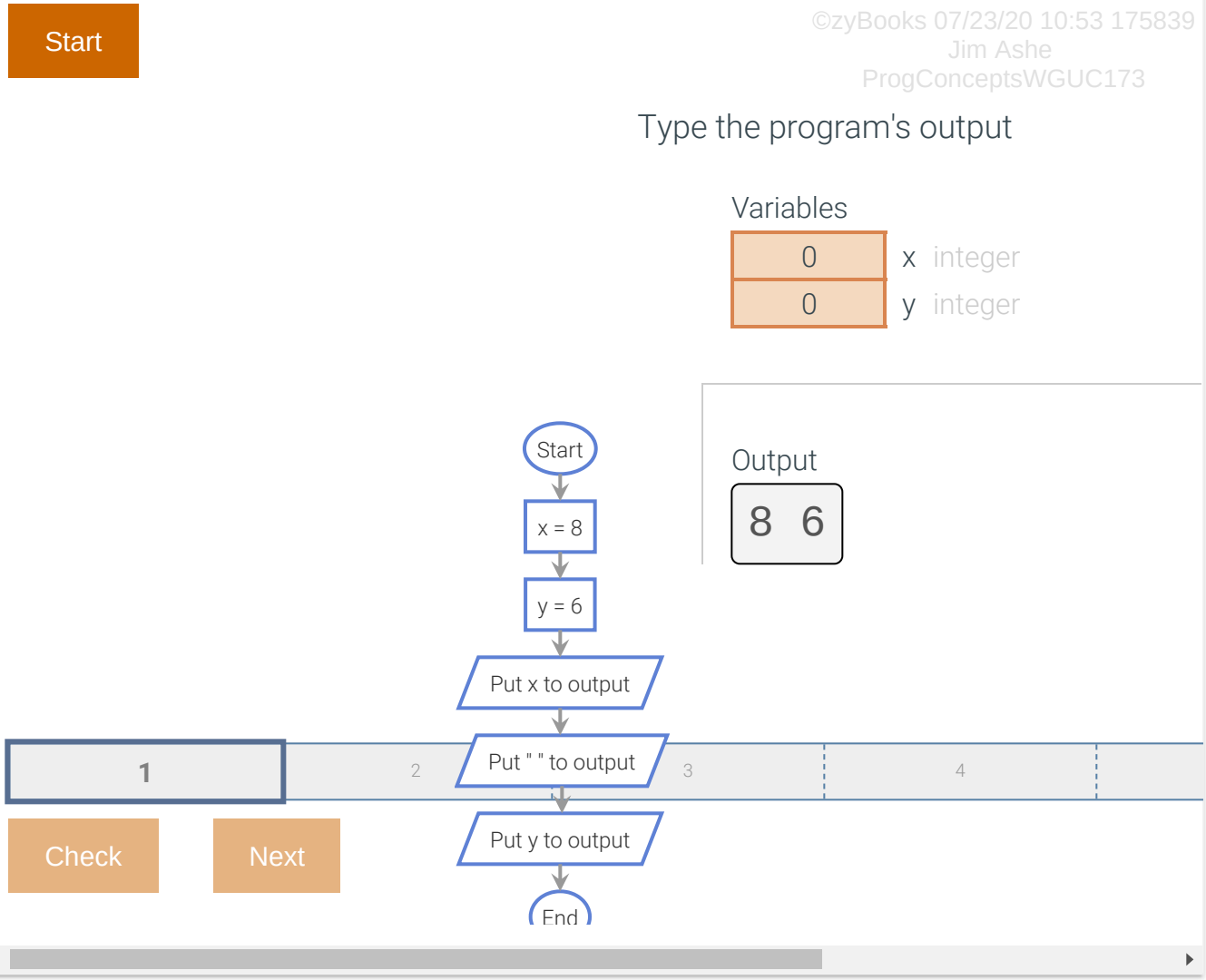


Check Show answer

3) Write a statement ending with - 1 that decreases variable flyCount's value by 1.

Check Show answer

CHALLENGE ACTIVITY 2.2.1: Enter the output of the variable assignments.



How was this section?

Provide feedback

2.3 Identifiers

Rules for identifiers

A name created by a programmer for an item like a variable or function is called an ***identifier***. An identifier must:

- be a sequence of letters (a-z, A-Z), underscores (_), and digits (0-9)
- start with a letter or underscore

Note that **_** is called an ***underscore***.

Identifiers are ***case sensitive***, meaning upper and lower case letters differ. So numCats and NumCats are different.

A ***reserved word*** (or ***keyword***) is a word that is part of the language, like integer, Get, or Put. A programmer cannot use a reserved word as an identifier. A list of reserved words appears at the end of this section.

PARTICIPATION ACTIVITY 2.3.1: Valid identifiers.

Which are valid identifiers?

1) numCars
   - Valid
   - Invalid

2) num_Cars1
   - Valid
   - Invalid

3) _numCars
   - Valid
   - Invalid

4) ___numCars
   - Valid
   - Invalid

5) 3rdPlace
   - Valid
   - Invalid

6) thirdPlace_
   - Valid

- Invalid

7) thirdPlace!

- Valid
- Invalid

8) Get

- Valid
- Invalid

9) very tall

- Valid
- Invalid

Style guidelines for identifiers

While various (crazy-looking) identifiers may be valid, programmers may follow identifier ***naming conventions***: A set of style guidelines defined by a company, team, teacher, etc., for naming variables.

Two common conventions for distinguishing words in an indentifier are:

- Camel case: ***Lower camel case*** abuts multiple words, capitalizing each word except the first, as in numApples or peopleOnBus.
- Underscore separated: Words are lowercase and separated by an underscore, as in num_apples or people_on_bus.

Neither convention is better. The key is to be consistent so code is easier to read and maintain.

Good practice is to create meaningful identifier names that self-describe an item's purpose. Good practice minimizes use of abbreviations in identifiers except for well-known ones like num in numPassengers. Programmers must strive to find a balance. Abbreviations make programs harder to read and can lead to confusion. Long variable names, such as averageAgeOfUclaGraduateStudent may be meaningful, but can make subsequent statements too long and thus hard to read.

PARTICIPATION ACTIVITY 2.3.2: Meaningful identifiers.

Choose the "best" identifier for a variable with the stated purpose, given the above discussion.

1) The number of students attending

UCLA.

- num
- numStdsUcla
- numStudentsUcla
- numberOfStudentsAttendingUcla

2) The size of an LCD monitor

- size
- sizeLcdMonitor
- s
- sizeLcdMtr

3) The number of jelly beans in a jar.

- numberOfJellyBeansInTheJar
- jellyBeansInJar
- nmJlyBnsInJr

4) The number of people in an elevator.

- n
- num
- numPeople

5) The number of tires on a car.

- numCar
- numTires
- numBowls

Table 2.3.1: zyFlowchart reserved words / function names.

Get Put to from input output	integer float array and or not	RaiseToPower SquareRoot RandomNumber SeedRandomNumbers

How was this section?

Provide feedback

2.4 Arithmetic expressions (general)

Basics

An ***expression*** is a combination of items, like variables, literals, operators, and parentheses, that evaluates to a value. Ex: 2 * (x + 1) is an expression. If x is 3, the expression evaluates to the value 8. Expressions are commonly used on the right side of an assignment statement, as in y = 2 * (x + 1).

A ***literal*** is a specific value in code, like 2. An ***operator*** is a symbol that performs a built-in calculation, like the operator + which performs addition. Common programming operators are shown below.

Table 2.4.1: Arithmetic operators.

Arithmetic operator	Description
+	The ***addition*** operator is **+**, as in x + y.
-	The ***subtraction*** operator is **-**, as in x - y. Also, the - operator is for ***negation***, as in -x + y, or x + -y.
*	The ***multiplication*** operator is ***, as in x * y.
/	The ***division*** operator is **/**, as in x / y.

PARTICIPATION ACTIVITY 2.4.1: Expressions.

Indicate which are valid expressions. x and y are variables.

1) x + 1

Valid

Not valid

2) 2 * (x - y)

Valid

Not valid

3) x

Valid

Not valid

4) 2

Valid

Not valid

5) 2x

Valid

Not valid

6) 2 + (xy)

Valid

Not valid

7) y = x + 1

Valid

Not valid

PARTICIPATION ACTIVITY 2.4.2: Capturing behavior with expressions.

Does the expression correctly capture the intended behavior?

1) 6 plus numItems:

```
6 + numItems
```

Yes

No

2) 6 times numItems:

`6 x numItems`

- Yes
- No

3) totDays divided by 12:

`totDays / 12`

- Yes
- No

4) 5 times i:

`5i`

- Yes
- No

5) The negative of userVal:

`-userVal`

- Yes
- No

6) n factorial

`n!`

- Yes
- No

Evaluation of expressions

An expression ***evaluates*** to a value, which replaces the expression. Ex: If x is 5, then x + 1 evaluates to 6, and y = x + 1 assigns y with 6.

An expression is evaluated using the order of standard mathematics, such order known in programming as ***precedence rules***, listed below.

Table 2.4.2: Precedence rules for arithmetic operators.

Convention	Description	Explanation
()	Items within parentheses are	In 2 * (x + 1), the x + 1 is evaluated

( )	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.	
+ -	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 2.4.3: Evaluating expressions.

Animation content:

```
x = 4
w = 2
expression is y = 3 * (x + 10 / 2)
10 / 2 is evaluated first, so 10 / 2 is 5
expression is 3 * (x + 5)
x + 5 is evaluated next, so 4 + 5 is 9
expression is 3 * 9
3 * 9 is evaluated, so 27
y = 27

The preferred method of making evaluation more clear is to add
parentheses, so the expression is rewritten as y = 3 * (x + (10 /
w))
```


Animation captions:

1. An expression like 3 * (x + 10 / w) evaluates to a value, using precedence rules. Items within parentheses come first, and / comes before +, yielding 3 * (x + 5).
2. Evaluation finishes inside the parentheses: 3 * (x + 5) becomes 3 * 9.
3. Thus, the original expression evaluates to 3 * 9 or 27. That value replaces the expression. So y = 3 * (x + 10 / w) becomes y = 27, so y is assigned with 27.
4. Many programmers prefer to use parentheses to make order of evaluation more clear when such order is not obvious.

PARTICIPATION ACTIVITY 2.4.4: Evaluating expressions and precedence rules.

Select the expression whose parentheses match the evaluation order of the original expression.

1) y + 2 * z
   - (y + 2) * z
   - y + (2 * z)

2) z / 2-x
   - (z / 2) - x
   - z / (2 - x)

3) x * y * z
   - (x * y) * z
   - x * (y * z)

4) x + 1 * y/2
   - ((x + 1) * y) / 2
   - x + ((1 * y) / 2)
   - x+ (1 * (y / 2))

5) x / 2 + y / 2
   - ((x / 2) + y) / 2
   - (x / 2) + (y / 2)

6) What is totCount after executing the following?

```
numItems = 5
totCount = 1 + (2 * numItems)  * 4
```

   - 44

41

Using parentheses to make the order of evaluation explicit

*A common error is to omit parentheses and assume an incorrect order of evaluation, leading to a bug. Ex: If x is 3, then 5 * x+1 might appear to evaluate as 5 * (3+1) or 20, but actually evaluates as (5 * 3) + 1 or 16 (spacing doesn't matter).*
*Good practice is to use parentheses to make order of evaluation explicit, rather than relying on precedence rules, as in: y = (m * x) + b, unless order doesn't matter as in x + y + z.*

Example: Calorie expenditure

A website lists the calories expended by men and women during exercise as follows (source):

Men: Calories = [(Age × 0.2017) — (Weight × 0.09036) + (Heart Rate × 0.6309) — 55.0969] × Time / 4.184

Women: Calories = [(Age × 0.074) — (Weight × 0.05741) + (Heart Rate × 0.4472) — 20.4022] × Time / 4.184

Below are those expressions written using programming notation:

caloriesMan = ((ageYears * 0.2017) - (weightPounds * 0.09036) + (heartBPM * 0.6309) - 55.0969) * timeMinutes / 4.184

caloriesWoman = ((ageYears * 0.074) - (weightPounds * 0.05741) + (heartBPM * 0.4472) - 20.4022)* timeMinutes / 4.184

PARTICIPATION ACTIVITY 2.4.5: Converting a formatted expression to a program expression.

Consider the example above. Match the changes that were made.

[ ] — × **Spaces in variable names**

	Replaced by ( )
	Single words
	-
	*

Reset

Modulo operator (%)

Note to instructors: zyFlowcharts supports the modulo operator (%), but this material does not teach the use of modulo operator.

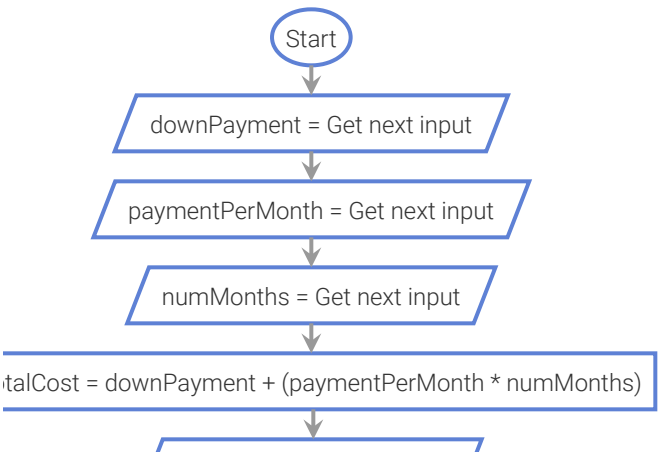
How was this section?

Provide feedback

2.5 Arithmetic expressions (integer)

Below is a simple program that includes an expression involving integers.

PARTICIPATION ACTIVITY 2.5.1: Expressions examples: Leasing cost. Full screen

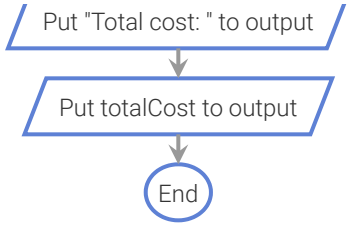


Variables

0	downPayment
0	paymentPerMonth
0	numMonths
0	totalCost

Input

500 300 60



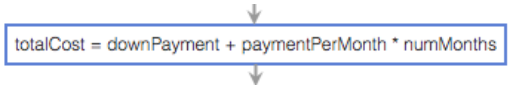
Output

ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 2.5.2: Simple program with an arithmetic expression.

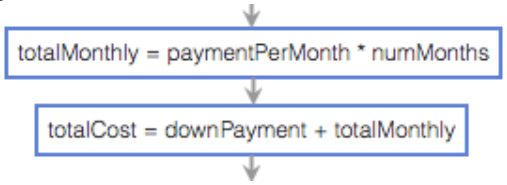
Consider the example above.

1) Would removing the parentheses, as shown below, have yielded the same result?



- Yes
- No

2) Would using two assignment statements, as shown below, have yielded the same result?



- Yes
- No

Style: Single space around operators

A good practice is to include a single space around operators for readability, as in numItems + 2, rather than numItems+2. An exception is minus used as negative, as in: xCoord = -yCoord. Minus (-) used as negative is known as ***unary minus***.

PARTICIPATION ACTIVITY 2.5.3: Single space around operators.

Retype each statement to follow the good practice of a single space around operators.

Note: If an answer is marked wrong, something differs in the spacing, spelling, capitalization, etc. This activity emphasizes the importance of such details.

1) `housesCity = housesBlock *10`

Check Show answer

2) `x = x1+x2+2`

Check Show answer

3) `numBalls=numBalls+1`

Check Show answer

4) `numEntries = (userVal+1)*2`

Check Show answer

No commas allowed

Commas are not allowed in an integer literal. So 1,333,555 is written as 1333555.

PARTICIPATION ACTIVITY 2.5.4: Expression in statements.

1) Is the following an error? Suppose an integer's maximum value is 2,147,483,647.

- Yes
- No

How was this section?

Provide feedback

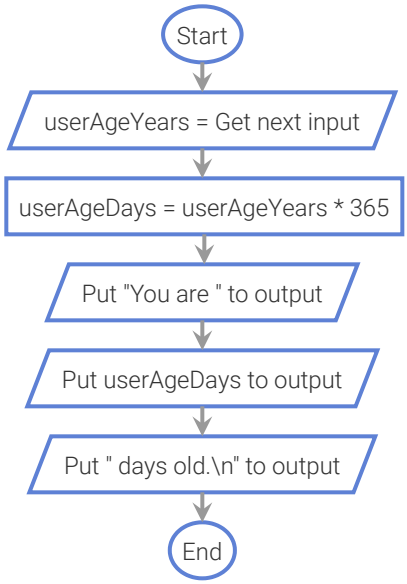
2.6 Example: Health data

Calculating user's age in days

This section presents an example program that computes various health related data based on a user's age using incremental development. ***Incremental development*** is the process of writing, compiling, and testing a small amount of code, then writing, compiling, and testing a small amount more (an incremental amount), and so on.

The initial program below calculates a user's age in days based on the user's age in years. The assignment statement `userAgeDays = userAgeYears * 365` assigns userAgeDays with the product of the user's age and 365, which does not take into account leap years.

PARTICIPATION ACTIVITY 2.6.1: Health data: Calculating user's age in days. **Full screen**



Variables

0	userAgeYears	integer
0	userAgeDays	integer

Input

19

Output

ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 2.6.2: Calculating user age in days.

1) Which variable is used for the user's age in years?

Check Show answer

2) If the input is 10, what will userAgeYears be assigned?

Check Show answer

3) If the user inputs 10, what is userAgeDays assigned?

Check Show answer

Considering leap years and calculating age in minutes

The program below extends the previous program by accounting for leap years when calculating the user's age in days. Since each leap year has one extra day, the statement `userAgeDays = userAgeDays + (userAgeYears / 4)` adds the number of leap years to userAgeDays. Note that the parentheses are not needed but are used to make the statement easier to read.

The program also computes and outputs the user's age in minutes.

PARTICIPATION ACTIVITY 2.6.3: Health data: Calculating user's age in days and minutes.

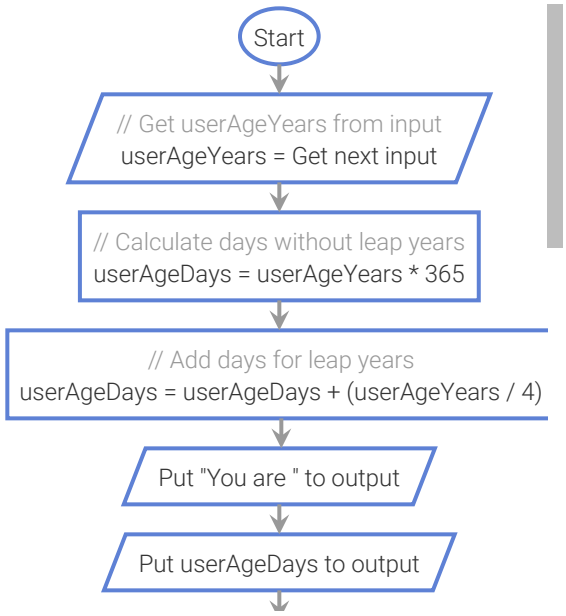
Full screen

Variables

0	userAgeYears	integer
0	userAgeDays	integer
0	userAgeMinutes	integer

Input

19



Output

_

ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 2.6.4: Calculating user age in days.

1) The expression `(userAgeYears / 4)` assumes a leap year occurs every four years?
   - True
   - False

2) The statement `userAgeDays = userAgeDays + (userAgeYears / 4)` requires parentheses to evaluate correctly.
   - True
   - False

3) If the user enters 20, what is userAgeDays after the first assignment statement?
   - 7300
   - 7305

4) If the user enters 20, what is userAgeDays after the second assignment statement?

- 7300
- 7305

Estimating total heartbeats in user's lifetime

The program is incrementally extended again to calculate the approximate number of times the user's heart has beat in his/her lifetime using an average heart rate of 72 beats per minutes.

PARTICIPATION ACTIVITY 2.6.5: Health data: Calculating total heartbeats lifetime.

Full screen

Variables

0	userAgeYears integer
0	userAgeDays integer
0	userAgeMinutes integer
0	totalHeartbeats integer
0	avgBeatsPerMinute integer

Input

19

Output

ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 2.6.6: Calculating user's heartbeats.

1) Which variable is used for the heart rate, which is measured in beats per minute?
   - heartRate
   - totalHeartbeats
   - avgBeatsPerMinute

2) If the user enters 10, what value is held in totalHeartbeats directly after the statement `userAgeDays = userAgeYears * 365`
   - 3650
   - 5258880
   - Unknown

How was this section?

Provide feedback

2.7 Floating-point numbers (float)

Floating-point (float) variables

A ***floating-point number*** is a real number, like 98.6, 0.0001, or -666.667. The term "floating-point" refers to the decimal point being able to appear anywhere ("float") in the number. A variable declared as type ***float*** stores a floating-point number.

A ***floating-point literal*** is a number with a fractional part, even if that fraction is 0, as in 1.0, 0.0, or 99.573. Good practice is to always have a digit before the decimal point, as in 0.5, since .5 might mistakenly be viewed as 5.

PARTICIPATION ACTIVITY 2.7.1: Variables of type float: Travel time example.

Animation content:

```
A Coral-Charts program follows:
float milesTraveled
float hoursToFly
float hoursToDrive

milesTraveled = Get next input
// Plane fly 500 MPH
hoursToFly = milesTraveled / 500.0
// Car drives 60 MPH
hoursToDrive = milesTraveled / 60.0
Put milesTraveled to output
Put " miles would take\n" to output
Put hoursToFly to output
Put " hours to fly or\n" to output
Put hoursToDrive to output
Put "hours to drive." to output

Variables in memory is as follows:
400.5 milesTraveled : float
0.801 hoursToFly : float
6.675 hoursToDrive : float

Input is as follows:
400.5

Output (screen) is as follows:
400.5 miles would take
0.801 hours to fly or
6.675 hours to drive.
```


Animation captions:

1. The program declares three float variables milesTraveled, hoursToFly, and hoursToDrive. Each is automatically assigned with 0.0.
2. The program gets input and assigns milesTraveled with 400.5.
3. The assignment statement reads milesTraveled (400.5), divides by 500.0, and assigns hoursToFly with the result of 0.801.
4. The assignment statement divides milesTraveled by 60.0 and assigns hoursToDrive with the result of 6.675.
5. The program outputs the time to fly or drive 400.5 miles.

PARTICIPATION ACTIVITY 2.7.2: Declaring and assigning float variables.

All variables are of type float and already-declared unless otherwise noted.

1) Assign a float variable named packageWeight with 7.1.

Check Show answer

2) Assign ballRadius with ballHeight divided by 2.0. Do not use the fraction 1.0 / 2.0. Instead, divide ballHeight directly by 2.0.

Check Show answer

3) Assign ballRadius with ballHeight multiplied by one half, namely (1.0 / 2.0). Use the parentheses around the fraction.

Check Show answer

PARTICIPATION ACTIVITY 2.7.3: Floating-point literals.

1) Which statement best assigns the float variable?

- currHumidity = 99%
- currHumidity = 99.0
- currHumidity = 99

2) Which statement best assigns the variable? Both variables are of type float.

- cityRainfall = measuredRain - 5
- cityRainfall = measuredRain - 5.0

3) Which statement best assigns the variable? cityRainfall is of type float.

- cityRainfall = .97
- cityRainfall = 0.97

Scientific notation

A system may print large or small floating-point values using scientific notation. Ex: If a float variable holds the value 299792458.0 (the speed of light in m/s), the value may be printed as 2.99792e+08. The printed value may have some rounding for conciseness, and the power of 10 makes the number's magnitude more apparent. The value held in the variable is unchanged.

Choosing a variable type (float vs. integer)

A programmer should choose a variable's type based on the type of value held.

- 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 values that are measured, like 98.6 degrees, 0.00001 meters, or -666.667 grams.
- Floating-point variables are also used when dealing with fractions of countable items, such as the average number of cars per household.

Note: Some programmers warn against using floating-point for money, as in 14.53 representing 14 dollars and 53 cents, because money is a countable item (reasons are discussed further in another section). Integers may be used to represent cents or to represent dollars when cents are not included, such as for an annual salary (e.g., 40000 dollars, which are countable).

PARTICIPATION ACTIVITY 2.7.4: Floating-point versus integer.

Choose the best variable type to represent each item.

1) The number of cars in a parking lot.

- float
- integer

2) The current temperature in Celsius.

- float
- integer

3) A person's height in centimeters.
   - float
   - integer

4) The number of hairs on a person's head.
   - float
   - integer

5) The average number of kids per household.
   - float
   - integer

Floating-point divide by zero

Dividing a nonzero floating-point number by zero results in ***infinity*** or ***-infinity***, depending on the signs of the operands. Printing a floating-point variable that holds infinity or -infinity outputs `Infinity` or `-Infinity`.

If the dividend and divisor in floating-point division are both 0, the division results in a "not a number". ***Not a number*** indicates an unrepresentable or undefined value. Printing a floating-point variable that is not a number outputs `NotANumber`.

PARTICIPATION ACTIVITY 2.7.5: Floating-point division by zero example.

PARTICIPATION ACTIVITY 2.7.6: Floating-point division.

Determine the result.

1) 13.0 / 3.0
   - 4
   - 4.333333
   - Positive infinity

2) 0.0 / 5.0

- 0.0
- Positive infinity
- Negative infinity

3) 12.0 / 0.0

- 12.0
- Positive infinity
- Negative infinity

4) 0.0 / 0.0

- 0.0
- Infinity
- Not a number

How was this section?

Provide feedback

2.8 Using math functions

Basics

Some programs require math operations beyond +, -, *, /, like computing a square root. zyFlowchart has a few built-in math operations, known as functions.

A ***function*** is a list of statements executed by invoking the function's name, with such invoking known as a ***function call***. Any function input values, or ***arguments***, appear within ( ), and are separated by commas if more than one. Below, the function SquareRoot is called with one argument, areaSquare. The function call evaluates to a value, as in SquareRoot(areaSquare) below evaluating to 7.0, which is assigned to sideSquare.

PARTICIPATION ACTIVITY 2.8.1: Using a math function.

Animation content:

A zyFlowchart program follows:

```
float areaSquare
float sideSquare

areaSquare = Get next Input
sideSquare = SquareRoot(areaSquare)
Put "Square root of " to output
Put areaSquare to output
Put " is " to output
Put sideSquare to output

Variables in memory is as follows:
49.0 areaSquare : float
7.0 sideSquare : float

Input is as follows:
49.0

Output is as follows:
Square root of 49.0 is 7.0
```


Animation captions:

1. The program computes the square's side length as the square root of the square's area.
2. A function is like a black box. The SquareRoot function takes an input value, and produces that value's square root.
3. Program outputs the square's area and side length.

Table 2.8.1: Math functions.

Function	Behavior	Example
SquareRoot(x)	Square root of x	SquareRoot(9.0) evaluates to 3.0.
RaiseToPower(x, y)	Raise x to power y: x^y	RaiseToPower(6.0, 2.0) evaluates to 36.0.
AbsoluteValue(x)	Absolute value of x	AbsoluteValue(-99.5) evaluates to 99.5.

PARTICIPATION ACTIVITY 2.8.2: Math functions.

1) SquareRoot(36.0) evaluates to _____

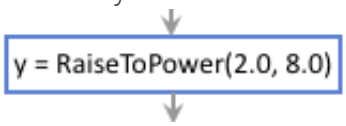
.

- 6.0
- 36.0

2) What is y?

- 9.0
- 81.0

3) What is y?



- 64.0
- 256.0

4) Is this a valid function call?

- Yes
- No

5) Is this a valid function call?

- Yes
- No

6) If w and x are float variables, is this a valid function call?

- Yes
- No

7) What is y?

- 8.0
- 16.0

Example: Mass growth

The example below computes the growth of a biological mass, such as a tree. If the growth rate is 5% per year, the program computes 1.05 raised to the number of years. A similar program could calculate growth of money given an interest rate.

PARTICIPATION ACTIVITY 2.8.3: Math function example: Mass growth.

PARTICIPATION ACTIVITY 2.8.4: Growth rate.

1) If initMass is 10.0, growthRate is 1.0 (100%), and yearsGrow is 3, what is finalMass?

```
finalMass = initMass * RaiseToPower(1.0 + growthRate, yearsGrow)
```

Check Show answer

PARTICIPATION ACTIVITY 2.8.5: Calculate Pythagorean theorem using math functions.

Select the three statements needed to calculate the value of x in the following:

$x = \sqrt{y^2 + z^2}$

For this exercise, calculate y^2 before z^2.

1) First statement is:

- temp1 = RaiseToPower(x , 2.0)
- temp1 = RaiseToPower(z , 3.0)
- temp1 = RaiseToPower(y , 2.0)

- temp1 = SquareRoot(y)

2) Second statement is:

- temp2 = SquareRoot(x , 2.0)
- temp2 = RaiseToPower(z , 2.0)
- temp2 = RaiseToPower(z)
- temp2 = x + SquareRoot(temp1 + temp2)

3) Third statement is:

- temp2 = SquareRoot(temp1 + temp2)
- x = RaiseToPower(temp1 + temp2, 2.0)
- x = SquareRoot(temp1) + temp2
- x = SquareRoot(temp1 + temp2)

Calls in arguments

Commonly a function call's argument itself includes a function call. Below, x^y is computed via RaiseToPower(x, y). The result is used in an expression that is an argument to another call, in this case to RaiseToPower() again: RaiseToPower(2.0, RaiseToPower(x, y) + 1).

PARTICIPATION ACTIVITY 2.8.6: Function call in an argument.

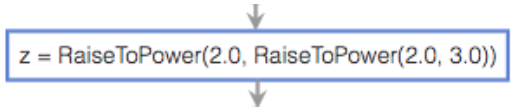
Animation captions:

1. x^y can be computed using RaiseToPower(x, y).
2. A function's argument can be an expression, including a call to another function. $2^{(x^y+1)}$ can be computed as RaiseToPower(2.0, RaiseToPower(x, y) + 1).
3. Upon execution, if x = 3.0 and y = 2.0, then RaiseToPower(x, y) is called and evaluates to 9.0. Next, RaiseToPower(2.0, 9.0 + 1) is called, yielding 1024.0.

PARTICIPATION ACTIVITY 2.8.7: Function calls in arguments.

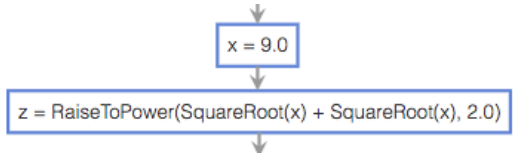
Type the ending value of z.

1)



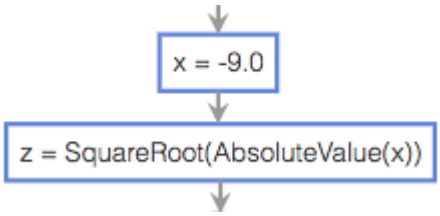
Check Show answer

2)



Check Show answer

3)



Check Show answer

How was this section?

Provide feedback

2.9 Random numbers

This section has been set as optional by your instructor.

Generating a random number

Some programs need to use a random number. Ex: A game program may need to roll dice, or a website program may generate a random initial password.

The ***RandomNumber()*** function is a built-in zyFlowchart function that takes two arguments, lowValue and highValue, and returns a random integer in the range

lowValue to highValue. Ex: RandomNumber(1, 10) returns a random integer in the range 1 to 10.

PARTICIPATION ACTIVITY 2.9.1: Generating random numbers in a range.

Animation captions:

1. Each call to RandomNumber() returns a random integer with the specified range. RandomNumber(1, 4) yields a random integer between 1 and 4.

PARTICIPATION ACTIVITY 2.9.2: Outputting three random integers.

PARTICIPATION ACTIVITY 2.9.3: Random number basics.

1) Which expression yields one of 5 possible values?
   - RandomNumber(1, 4)
   - RandomNumber(0, 5)
   - RandomNumber(1, 5)

2) Which expression yields one of 100 possible values?
   - RandomNumber(0, 100)
   - RandomNumber(0, 99)

3) What is the largest possible value returned by RandomNumber(0, 25)?
   - 0
   - 25

4) Which expression would best mimic the random outcome of flipping a coin?
   - RandomNumber(0, 0)
   - RandomNumber(0, 1)
   - RandomNumber(0, 2)

PARTICIPATION ACTIVITY 2.9.4: Generating random integers in a specific range.

1) Goal: Random integer from the 6 possible values 0 to 5.
RandomNumber(0, ____);

Check Show answer

2) Goal: Random integer from 1 to 5.
RandomNumber(____)

Check Show answer

3) How many values exist in the range 10 to 15?

Check Show answer

4) How many values exist in the range 10 to 100?

Check Show answer

5) Goal: Random integer in the range -20 to 20.
RandomNumber(____)

Check Show answer

6) How many values are in the range -5 to 5?

Check Show answer

Example: Moving students's seats

The following program randomly moves a student from one seat to another seat in a lecture hall, perhaps to randomly move students before an exam. The seats are in 20 rows numbered 1 to 20. Each row has 30 seats (columns) numbered 1 to 30. The student should be moved from the left side (columns 1 to 15) to the right side (columns 16 to 30).

PARTICIPATION ACTIVITY 2.9.5: Randomly moving a student from one seat to another.

PARTICIPATION ACTIVITY 2.9.6: Random integer example: Moving seats.

Consider the above example.

1) The row is chosen using RandomNumber(1, 20). The 20 is because 20 _____ exist.
   - rows
   - columns

2) The column for the left is chosen using RandomNumber(1, 15). The 15 is used because the left half of the hall has _____ columns.
   - 15
   - 30

3) The column for the right could have been chosen using RandomNumber(1, 15) + 15.
   - True
   - False

Pseudo-random

The integers generated by RandomNumber() are known as pseudo-random. "Pseudo" means "not actually, but having the appearance of". Internally, the RandomNumber() function has an equation to compute the next "random" integer from the previous one, (invisibly) keeping track of the previous one. For the first call to RandomNumber(), no previous random integer exists, so the function uses a built-in integer known as the ***seed***. zyFlowchart automatically seeds the

pseudo-random number generator with a number based on the current time. Since, the time is different for each program run, the program will get a unique sequence.

Reproducibility is important for testing some programs. A programmer can specify the seed using the function ***SeedRandomNumbers()***, as in SeedRandomNumber(10) or SeedRandomNumber(99). Note that the seeding should only be done once in a program, before the first call to RandomNumber().

PARTICIPATION ACTIVITY 2.9.7: Using the same seed for each program run.

PARTICIPATION ACTIVITY 2.9.8: Seeding a pseudo-random number generator.

1) By starting a program with SeedRandomNumbers(15), calls to RandomNumber() will yield a different integer sequence for each program run.
   - ○ True
   - ○ False

2) If a program does not make a call to SeedRandomNumbers(), calls to RandomNumber() will yield a different integer sequence for each successive program run.
   - ○ True
   - ○ False

3) RandomNumber() is known as generating a "pseudo-random" sequence of values because the sequence begins repeating itself after about 20 numbers.
   - ○ True
   - ○ False

©zyBooks 07/23/20 10:53 175839
Jim Ashe
ProgConceptsWGUC173

How was this section?

Provide feedback

2.10 Integer division

Division: Integer rounding

When the operands of / are both integers, the operator performs integer division, which does not generate any fraction. The / operator performs floating-point division if at least one operand is a floating-point type.

PARTICIPATION ACTIVITY 2.10.1: Integer division does not generate any fraction.

Animation captions:

1. If both operands of / are integers, the operator performs integer division: No fractional part is generated. Thus 10 / 4 is 2, not 2.5. And 3 / 4 is 0, not 0.75.
2. Programmers may forget, causing strange logic errors. (1 / 2) * b * h is always (0) * b * h or 0. And c * (9 / 5) + 32 is always c * (1) + 32.
3. The same applies for integer variables. No fraction is generated for y = w / x if w and x are integers, even if y is a floating-point type.
4. If at least one operand of / is a floating-point type, then floating-point division occurs. So if integer w = 10 and float x = 4.0, then w / x is 2.5.

PARTICIPATION ACTIVITY 2.10.2: Integer division and modulo.

Determine the result. Some expressions only use literals to focus attention on the operator, but in practice most expressions include variables.

1) `13 / 3`

Check Show answer

2) `4 / 9`

Check Show answer

3) `(5 + 10 + 15) * (1 / 3)`

Check Show answer

4) `x / y` where integer x = 10 and integer y = 4.

Check Show answer

5) `10 / 4.0`

Check Show answer

6) `x / y` where integer x = 10 and float y = 4.0.

Check Show answer

Division: Divide by 0

For integer division, the second operand of / or % must never be 0, because division by 0 is mathematically undefined. A ***divide-by-zero error*** occurs at runtime if a divisor is 0, causing a program to terminate.

PARTICIPATION ACTIVITY 2.10.3: Divide-by-zero example: Compute salary per day.

PARTICIPATION ACTIVITY 2.10.4: Integer division and modulo.

Determine the result. Type "Error" if the program would terminate due to divide-by-zero. Some expressions only use literals to focus attention on the operator, but most practical expressions include variables.

1) 100 / 2

Check Show answer

2) 100 * (1 / 2)

Check Show answer

3) 100 * 1 / 2

Check Show answer

4) 100 / (1 / 2)

Check Show answer

5)
```
x = 2;
y = 5;
z = 1 / (y - x - 3);
```

Check Show answer

How was this section?

Provide feedback

2.11 Type conversions

Type conversions

A calculation sometimes must mix integer and floating-point numbers. For example, given that about 50.4% of human births are males, then `0.504 * numBirths` calculates the number of expected males in numBirths births. If numBirths is an integer variable (integer because the number of births is countable), then the expression combines a floating-point and integer.

A ***type conversion*** is a conversion of one data type to another, such as an integer to a float. zyFlowchart automatically performs several common conversions between integer and float types, and such automatic conversion is known as ***implicit conversion***.

- For an arithmetic operator like + or *, if either operand is a float, the other is automatically converted to float, and then a floating-point operation is performed.
- For assignments, the right side type is converted to the left side type.

integer-to-float conversion is straightforward: 25 becomes 25.0.

float-to-integer conversion just drops the fraction: 4.9 becomes 4.

PARTICIPATION ACTIVITY 2.11.1: Implicit type conversion: integer-to-float.

Animation captions:

1. 0.504 is a floating-point literal. numBirths is an integer variable. zyFlowchart sees "float * integer" and performs an implicit integer-to-float type conversion.
2. If numBirths is 316, 316 is first converted 316.0.
3. Then, the program computes 0.504 * 316.0 yielding 159.264. expectedMales is a float variable and is assigned with that result.

PARTICIPATION ACTIVITY 2.11.2: Implicit conversions among float and integer.

Type the value of the expression given integer numItems = 5. For any floating-point answer, type answer to tenths. Ex: 8.0, 6.5, or 0.1.

1) 3.0 / 1.5

Check Show answer

2) 3.0 / 2

Check Show answer

3) (numItems + 10) / 2

Check Show answer

4) (numItems + 10) / 2.0

Check Show answer

PARTICIPATION ACTIVITY 2.11.3: Implicit conversions among float and integer with variables.

Type the value held in the variable after the assignment statement, given integer numItems = 5, float itemWeight = 0.5. For any floating-point answer, type answer to tenths. Ex: 8.0, 6.5, or 0.1

1) someFloatVar = itemWeight * numItems (someFloatVar is type float).

Check Show answer

2) someIntVar = itemWeight * numItems (someIntVar is type integer).

Check Show answer

Type casting

A programmer sometimes needs to explicitly convert an item's type. Ex: If a program needs a floating-point result from dividing two integers, then at least one of the integers needs to be converted to a float so floating-point division is performed. Otherwise, integer division is performed, evaluating to only the quotient and ignoring the remainder.

A ***type cast*** converts a value of one type to another type. A programmer can type cast an integer to float by multiply the integer by the float literal 1.0. Ex: If myIntVar is 7, then `myIntVar * 1.0` converts integer 7 to float 7.0.

The program below casts the numerator to float so floating-point division is performed.

PARTICIPATION ACTIVITY 2.11.4: Using type casting to obtain floating-point division.

PARTICIPATION ACTIVITY 2.11.5: Type casting.

Determine the resulting type for each expression. Assume numSales1, numSales2, and totalSales are integer variables.

1) (numSales1 + numSales2) / 2
   - integer
   - float

2) (1.0 * (numSales1 + numSales2)) / 2
   - integer
   - float

3) (numSales1 + numSales2) / totalSales
   - integer
   - float

4) (numSales1 + numSales2) / (1.0 * totalSales)
   - integer
   - float

Common errors

A common error is to accidentally perform integer division when floating-point division was intended. The program below undesirably performs integer division rather than floating-point division.

PARTICIPATION ACTIVITY 2.11.6: Common error: Forgetting to cast results in integer division.

Another common error is to cast the entire result of integer division, rather than the operands, thus not obtaining the desired floating-point division.

PARTICIPATION ACTIVITY 2.11.7: Common error: Casting final result instead of operands.

Animation captions:

1. The programmer wants to use floating-point division to compute the average of midtermScore (an integer) and finalScore (an integer).
2. (midtermScore + finalScore) is evaluated first. If midtermScore is 90 and finalScore is 85, the expression evaluation to (90 + 85) or 175.
3. Next, 175 / 2 is evaluated. Both operands are integers, so integer divisions is performed, yielding 87.
4. Multiplying by 1.0 converts 87 to 87.0. Casting the result of integer division does not perform the desired floating-point division.

PARTICIPATION ACTIVITY 2.11.8: Type casting.

1) Given integer aVal is 10 and integer bVal is 4, which yields 2.5?
   - (1 * aVal) / (1 * bVal)
   - (1.0 * aVal) / (1.0 * bVal)
   - 1.0 * (10 / 4)

2) Given integer aVal is 15 and integer bVal is 4, which does NOT yield 3.75?
   - (1.0 * aVal) / (1.0 * bVal)
   - (1.0 * aVal) / bVal
   - aVal / (1.0 * 4)
   - 1.0 * (aVal / bVal)

3) Given aCnt, bCnt, and cCnt are integer variables, which variable must be cast to a float for the expression `(aCnt * bCnt) / cCnt` to evaluate to a float value?
   - None
   - All variables
   - Only one variable

How was this section?

Provide feedback

2.12 Modulo operator

The basic arithmetic operators include not just +, -, *, /, but also %. The ***modulo operator*** (**%**) evaluates to the remainder of the division of two integer operands. Ex: 23 % 10 is 3.

Examples:

- 24 % 10 is 4. Reason: 24 / 10 is 2 with remainder 4.
- 50 % 50 is 0. Reason: 50 / 50 is 1 with remainder 0.
- 1 % 2 is 1. Reason: 1 / 2 is 0 with remainder 1.
- 10 % 4.0 is not valid. "Remainder" only makes sense for integer operands.

PARTICIPATION ACTIVITY 2.12.1: Division and modulo example: Minutes to hours/minutes.

PARTICIPATION ACTIVITY 2.12.2: Modulo.

Determine the result. Type "Error" if appropriate. Only literals appear in these expressions to focus attention on the operators; most practical expressions include variables.

1) `50 % 2`

Check Show answer

2) `51 % 2`

Check Show answer

3) `78 % 10`

Check Show answer

4) `596 % 10`

Check Show answer

5) `100 % (1 / 2)`

Check Show answer

6) `100.0 % 40`

Check Show answer

CHALLENGE ACTIVITY 2.12.1: Enter the output of the integer expressions.

How was this section?

Provide feedback

2.13 Data types

A programmer should choose a data type that best matches the value being held or represented. The following describes common data types used in most programming languages.

Table 2.13.1: Common data types.

Data type	Description
Integer	An integer is a whole number value, like 1, 999, 0, or -25.
Float	A float is a floating-point number, which is a real number, like 98.6, 0.0001, or -666.667.
Character	A character is a single letter, like 'A', 'p', or '%'.
String	A string is a sequence of characters, like "Hello" or "The forecast for today is sunny with highs of 75F.".

Boolean	A Boolean refers to a quantity that has only two possible values, true or false.
Array	An array is an ordered list of items of a given data type, like an array of integers or an array of floats.

zyFlowchart supports variables of integer, float, and array data type, string literals that can be put to output, and Boolean values resulting for equality, relational, or logical operators.

PARTICIPATION ACTIVITY 2.13.1: Data types.

Determine which data type is best for the stated purpose.

Boolean **Float** **Float array** **String array** **Integer**

	A program needs to keep track of the number of TVs available for sale.
	A program needs to store a list of people waiting to buy concert tickets.
	A program needs to store the amount of rain fall in inches for the past day.
	A program needs to store the forecasted high temperatures for the next 10 days.
	A program needs to keep track of whether a person's airline ticket is confirmed or not.

Reset

How was this section?

Provide feedback

2.14 Constants

A ***constant*** is a named value item that holds a value that cannot change. Constants are commonly used in programs to hold the value of mathematical or physical constants, such as Pi, the speed of light, or kilograms per pound. Constants can also be used for any value that should not change during the program's execution.

A good practice is to minimize the use of literal numbers in code by using constants to improve code readability. Ex: newPrice = origPrice - 5 is less clear than newPrice = origPrice - PRICE_DISCOUNT, where PRICE_DISCOUNT is a constant.

A common convention, or good practice, is to name constants using upper case letters with words separated by underscores, to make constants clearly visible.

PARTICIPATION ACTIVITY 2.14.1: Constants: Lightning distance estimation.

Animation content:

```
A Coral-Charts program follows:
float SOUND_SPEED
integer SEC_PER_HOUR

// Get seconds between lightning and thunder
secBetween = Get next input
timeInHours = secBetween / SEC_PRE_HOUR
distInMiles = SOUND_SPEED * timeInHours
Put "Miles from lightning strike: " to output
Put distInMiles to output
```



```
Constants in memory is as follows:
761.201 SOUND_SPEED: float
3600 SEC_PER_HOUR: integer

Variables in memory is as follows:
7.0 secBetween: float
0.0019 timeInHours: float
```

```
1.4801 distInMiles float

Input is as follows:

Output (screen) is as follows:
Miles from lightning strike: 1.4801
```

Animation captions:

1. A constant is a named value item that holds a value that cannot change. SOUND_SPEED is a float constant holding the speed of sound at sea level in miles/hour.
2. SEC_PER_HOUR is an integer constant holding the number of seconds in one hour.
3. The program gets the time between seeing lightning and hearing thunder, and then estimates how far away the lightning strike was.
4. Constants can be used in expressions. SEC_PER_HOUR is used to calculate the time in hours, and SOUND_SPEED is used to calculate the distance in miles.

PARTICIPATION ACTIVITY 2.14.2: Constants and variables.

Indicate if the following values should be held in a variable or constant.

1) The value for Pi.
   - ○ Variable
   - ○ Constant

2) User input for a weight measurement in kilograms.
   - ○ Variable
   - ○ Constant

3) A value for fixed tax rate of 7.5.
   - ○ Variable
   - ○ Constant

4) Income tax calculated for a user's annual pay with a fixed tax rate of 7.5.
   - ○ Variable
   - ○ Constant

How was this section?

Provide feedback

2.15 Pseudocode: Variables and assignments

A programmer can use pseudocode to describe a flowchart's variables and statements by:

1. Declaring each variable's type and name, one variable declaration per line. Ex: `integer userAge` declares the variable named userAge of type integer.
2. Listing each program statement in order, one statement per line.

To provide separation between different parts of a program, variable declarations and program statements are often separated by a blank line, which is considered whitespace and not part of the program structure.

PARTICIPATION ACTIVITY 2.15.1: Pseudocode: variables and assignments.

Animation content:

```
A zyFlowchart program shown graphically on the left and in
pseudocode on the right:
float areaSquare
float sideSquare
areaSquare = Get next input
sideSquare = SquareRoot(areaSquare)
Put "Square root of " to output
Put areaSquare to output
Put "is " to output
Put sideSquare to output

Variables in memory is as follows:
0.0 areaSquare: float
0.0 sideSquare: float
```


Animation captions:

1. A program's variables are declared by specifying the type and name, one per line.

2. A program's statements are listed in order of execution, one statement per line. A blank line separates the variable declarations and statements for readability, but is not required in pseudocode.

PARTICIPATION ACTIVITY 2.15.2: Pseudocode: Variables and assignments.

Given the pseudocode below:

```
float milesTraveled
float hoursToFly
float hoursToDrive

milesTraveled = Get next input

// Plane flies 500 MPH
hoursToFly = milesTraveled / 500.0

// Car drives 60 MPH
hoursToDrive = milesTraveled / 60.0

Put milesTraveled to output
Put " miles would take\n" to output
Put hoursToFly to output
Put " hours to fly or\n" to output
Put hoursToDrive to output
Put "hours to drive." to output
```

1) How many variable are declared?

- 1
- 2
- 3

2) Which statement executes first?

- float milesTraveled
- milesTraveled = Get next input
- Put "hours to drive." to output

3) Which statement executes second?

- float hoursToFly
- // Plane flies 500 MPH
- hoursToFly = milesTraveled / 500.0

How was this section?

Provide feedback

2.16 Array concept (general)

A typical variable stores one data item, like the number 59 or the character 'a'. Instead, sometimes a *list* of data items should be stored. Ex: A program recording points scored in each quarter of a basketball game needs a list of 4 numbers. Requiring a programmer to declare 4 variables is annoying; 200 variables would be ridiculous. An ***array*** is a special variable having one name, but storing a list of data items, with each item being directly accessible. Some languages use a construct similar to an array called a ***vector***. Each item in an array is known as an ***element***.

PARTICIPATION ACTIVITY 2.16.1: Sometimes a variable should store a list, or array, of data items.

Animation content:

```
numPlayers
12

pointerPerQuarter
element 0 contains 22
element 1 contains 19
element 2 contains 12
element 3 contains 28

How many points in 4th quarter?
pointsPerQuarter[3] is 28
```

Animation captions:

1. A variable usually stores just one data item.
2. Some variables should store a list of data items, like variable pointsPerQuarter that stores 4 items.
3. Each element is accessible, like the element numbered 3.

You might think of a normal variable as a truck, and an array variable as a train. A truck has just one car for carrying "data", but a train has many cars, each of which can carry data.

Figure 2.16.1: A normal variable is like a truck, whereas an array variable is like a train.


Source: Truck (Ian Britton / freefoto.com), train (Ian Britton / freefoto.com)

In an array, each element's location number is called the ***index***; myArray[2] has index 2. An array's key feature is that the index enables direct access to any element, as in myArray[2]; different languages may use different syntax, like myArray(3) or myVector.at(3). In many languages, indices start with 0 rather than 1, so an array with 4 elements has indices 0, 1, 2, and 3.

PARTICIPATION ACTIVITY 2.16.2: Array basics.

Array peoplePerDay has 365 elements, one for each day of the year. Valid accesses are peoplePerDay[0], [1], ..., [364].

1) Which assigns element 0 with the value 250?

- peoplePerDay[250] = 0
- peoplePerDay[0] = 250
- peoplePerDay = 250

2) Which assigns element 1 with the value 99?

- peoplePerDay[1] = 99
- peoplePerDay[99] = 1

3) What is the value of peoplePerDay[8]?

```
peoplePerDay[9] = 5
peoplePerDay[8] = peoplePerDay[9]
- 3
```

- 8
- 5
- 2

4) Assume N is initially 1. What is the value of peoplePerDay[2]?

```
peoplePerDay[N] = 15
N = N + 1
peoplePerDay[N] = peoplePerDay[N - 1] * 3
```

- 15
- 2
- 45

PARTICIPATION ACTIVITY 2.16.3: Arrays with element numbering starting with 0.

Array scoresList has 10 elements with indices 0 to 9, accessed as scoresList[0] to scoresList[9].

1) Assign the first element in scoresList with 77.

Check Show answer

2) Assign the second element in scoresList with 77.

Check Show answer

3) Assign the last element with 77.

Check Show answer

4) If that array instead has 100 elements, what is the last element's index?

Check Show answer

Check Show answer

5) If the array's last index was 499, how many elements does the array have?

Check Show answer

PARTICIPATION ACTIVITY 2.16.4: Update the array's data values.

How was this section?

Provide feedback

2.17 Arrays

Array declarations and accessing elements

A programmer commonly needs to maintain a list of items, just as people often maintain lists of items like a grocery list or a course roster. An ***array*** variable is an ordered list of items of a given data type and size. Each item in an array is called an ***element***. For array x, each element is accessed as x[0], x[1], ... In an array access, the number in brackets is called the ***index*** of that element. The first array element is at index 0.

Some relevant terminology

- *[ ] are* ***brackets***
- *{ } are* ***braces***
- *( ) are* ***parentheses***

To contrast with array variables, a single-item (non-array) variable is called a ***scalar*** *variable.*

The program below declares an integer array named itemCounts, having 3 elements. The program assigns the first element itemCounts[0] with an input value, the next element with 99, and the third element with itemCounts[0] * 2.

Note below that in Coral, an array stores a ***size*** attribute, indicating the number of array elements.

PARTICIPATION ACTIVITY 2.17.1: A simple array example.

PARTICIPATION ACTIVITY 2.17.2: Array basics.

Consider the array example above.

1) How many elements does the array declaration create?
   - 0
   - 2
   - 3

2) itemCounts[1] is assigned with what value?
   - 1
   - 99

3) If the input value is 25, itemCounts[2] is assigned with what value?
   - 25
   - 50
   - 52

4) Which is a valid assignment? Assume x is an integer variable.
   - itemCounts = x
   - x = itemCounts + 1
   - x = itemCounts[2] + 1

5) What is the proper way to access the *first* element?

- itemCounts[1]
- itemCounts[0]

6) The array had size 3. What is the index of the *last* element of the array?
   - 2
   - 3
   - 4

Indices

New programmers often need practice to remember that array indices start with 0 rather than 1. The below activity may help.

PARTICIPATION ACTIVITY 2.17.3: Select the index shown.

Using an expression for an array index

A powerful aspect of arrays is that the index is an expression. Ex: userNums[i] uses the value held in the integer variable `i` as the index. As such, an array is useful to easily lookup the Nth item in a list.

An array's index must be an integer type. The array index cannot be a floating-point type, even if the value is 0.0, 1.0, etc.

The program below allows a user to print the age of the Nth oldest known person to have ever lived. The program quickly accesses the Nth oldest person's age using `oldestPeople[nthPerson - 1]`. Note that the index is `nthPerson - 1` rather than just `nthPerson` because an array's indices start at 0, so the 1st age is at index 0, the 2nd at index 1, etc.

PARTICIPATION ACTIVITY 2.17.4: A Nth oldest person program (age source: Wikipedia.org).

PARTICIPATION ACTIVITY 2.17.5: Nth oldest person program.

1) In the program above, what is the purpose of this decision statement:

```
((nthPerson >= 1) and (nthPerson
<= 5))
```

- To avoid assigning too large a value because nthPerson's data type can only store values from 1 to 5.
- To ensure only valid array elements are accessed because the array oldestPeople only has 5 elements.

PARTICIPATION ACTIVITY 2.17.6: Array accesses.

1) Assign variable x with the value stored at index 8 of array myVals.

Check Show answer

2) Assign the second element of array myVals with the value 555.

Check Show answer

3) Assign myVals array element at the index held in currIndex with the value 777.

Check Show answer

4) Assign tempVal with the myVals array element at the index one after the value held in variable i.

Check Show answer

Arrays and loops

A key advantage of arrays becomes evident when used with loops. The program below uses a loop to allow a user to enter 5 integer values, storing those values in array userVals, and then outputting those 5 values.

The loop makes use of the array's size attribute, accessed via userVals.size, which in this case has value 5.

PARTICIPATION ACTIVITY 2.17.7: Arrays and loops.

PARTICIPATION ACTIVITY 2.17.8: Array with loops.

Consider the program above.

1) How many times does each loop iterate?
   - 1
   - 5
   - Unknown

2) Which item should be changed to allow the user to enter 100 values?
   - integer array(5) userVals
   - i = i + 1

CHALLENGE ACTIVITY 2.17.1: Enter the output for the array.

How was this section?

Provide feedback

©zyBooks 07/23/20 10:53 175839
Jim Ashe
ProgConceptsWGUC173

2.18 Programming fundamentals summary

This chapter's key points included:

- A variable declaration declares a new variable, specifying the variable's name and type.
- An assignment statement assigns the variable on the left-side of the = with the current value of the right-side expression.
- An expression is a combination of items, like variables, literals, operators, and parentheses, that evaluates to a value.
- A name created by a programmer for an item like a variable or function is called an identifier, which must follow certain rules to be valid. Programmers typically follow identifier naming conventions that are defined by their company, team, teacher, etc.
- An expression is evaluated using precedence rules that follow the evaluation order of standard mathematics.
- Incremental development is the process of writing and and testing a small amount of code, then writing and testing a small amount more (an incremental amount), and so on.
- A variable declared as type float stores a floating-point number, which is a real number, like 98.6, 0.0001, or -666.667.
- A programmer should choose a variable's type based on the type of value held. Integer variables are typically used for values that are counted. Floating-point variables are typically used for values that are measured or when dealing with fractions of countable items, such as the average number of cars per household.
- Programming languages typically have built-in functions to perform common operations needed by programmers, such as performing mathematical operations like square root or raising a number to a power.
- A function is a list of statements executed by invoking the function's name, with such invoking known as a function call.
- Programming languages typically have built-in functions for generating random numbers. The integers generated by a random number generator are known as pseudo-random. "Pseudo" means "not actually, but having the appearance of". Internally, the RandomNumber() function has an equation to compute the next "random" integer from the previous one.
- When the operands of / are both integers, the operator performs integer division, which does not generate any fraction.
- For integer division, the second operand of / or % must never be 0, because division by 0 is mathematically undefined. A divide-by-zero error occurs at runtime if a divisor is 0, causing a program to terminate.
- A type conversion is a conversion of one data type to another, such as an integer to a float.
- zyFlowchart, and other programming languages, automatically performs several common conversions between integer and float types, and such automatic conversion is known as implicit conversion.
- If a programmer needs to explicitly convert an item's type, the programmer can use a type cast to converts value of one type to another type.
- The modulo operator (%) evaluates to the remainder of the division of two integer operands.

- A constant is a named value item that holds a value that cannot change. Constants are commonly used in programs to hold the value of mathematical constants or a value that should not change during the program's execution.
- An array variable stores multiple items, each accessible using an index. In contrast, a scalar variable stores just one item.
- Array indices start from 0, not 1. An index may be represented as an expression.

PARTICIPATION ACTIVITY 2.18.1: Variables/assignments summary.

1) Which statement assigns itemSavings with the difference of normalPrice and salePrice.
   - itemSavings = normalPrice - salePrice
   - itemSavings = salePrice + normalPrice
   - itemSavings = normalPrice * salePrice

2) The weekly sales for a sales person was 14 cars. Which operation should be used to compute the average daily sales, which is 2.0 cars per day?
   - Addition
   - Multiplication
   - Division

3) A variable should hold the number of people attending an event. What data type should the variable be?
   - Integer
   - Float
   - Boolean

4) Given x = 10 and y = 4. What value does the expression (x - y) * 2 evaluate to?
   - 2
   - 6
   -

12

5) What is the role of * in the expression x + z * 10?

- Assignment operator
- Arithmetic operator
- Relational operator

6) Given float y = 15.5. Which expression evaluates to 4.0?

- y - 10.0
- (y + 0.5) / 2.0
- (y + 0.5) / 4.0

7) A program determines if the number of people admitted to a concert venue exceeds the maximum seats of 1500. How should the item to hold the maximum seats be declared?

- Constant integer maxSeats
- Variable integer numPeople
- Constant float maxTicketPrice

8) Which data type should be used to hold the value 210.25?

- Integer
- Float
- String

PARTICIPATION ACTIVITY 2.18.2: Arrays.

1) What is the best declaration type for a person's height?

- scalar
- array

2) What is the best declaration type for maintaining a person's weight for every month of the year?

- scalar
- array

3) What is the best declaration type for a list of player numbers on a soccer team?

- scalar
- array

How was this section?

Provide feedback

©zyBooks 07/23/20 10:53 175839
Jim Ashe
ProgConceptsWGUC173

3.1 Branches

Branches

In a program, a ***branch*** is a sequence of statements only executed under a certain condition. In zyFlowchart, a ***decision*** creates two branches: If the decision's expression is true, the first branch executes, else the second branch executes. Afterwards, the branches rejoin. zyFlowchart uses a diamond symbol for a decision.

PARTICIPATION ACTIVITY 3.1.1: Branches: Hotel rate example.

Animation content:

```
A zyFlowchart program follows:
integer hotelRate
integer userAge

hotelRate = 155
userAge = Get next input
if userAge > 65
   hotelRate = hotelRate - 20
Put "Your rate: " to output
Put hotelRate to output

Variables in memory is as follows:
135 hotelRate: integer
68 userAge: integer

Input is as follows:
68

Output (screen) is as follows:
Your rate: 135
```


Animation captions:

1. A decision leads to two program branches. If the expression is true, the first branch executes. Else, the second branch executes.

2. If userAge is 68, then 68 > 65 is true so the first branch executes, which discounts hotelRate.
3. Execution rejoins the other branch, and continues with subsequent statements, outputting 135. If userAge were instead 50, the output would be 155.

A decision and its two branches are often called ***if-else*** branches, because IF the decision's expression is true then the first branch executes, ELSE the second branch executes.

In the example above, the else (false) branch had no statements. A decision whose false branch has no statements is often just called an ***if*** branch.

PARTICIPATION ACTIVITY 3.1.2: Branches.

Consider the hotel rate example above.

1) If userAge is 20, does the true or false branch execute?
   - ○ True branch
   - ○ False branch

2) If userAge is 20, does the executed branch update hotelRate?
   - ○ Yes
   - ○ No

3) If userAge is 20, what hotel rate does the program output?
   - ○ 155
   - ○ 135

4) If userAge is 70, what hotel rate does the program output?
   - ○ 155
   - ○ 135

5) Do the last two statements always execute for any value of userAge?
   - ○ Yes
   - ○ No

6) A decision and its two branches are often called _____ branches.

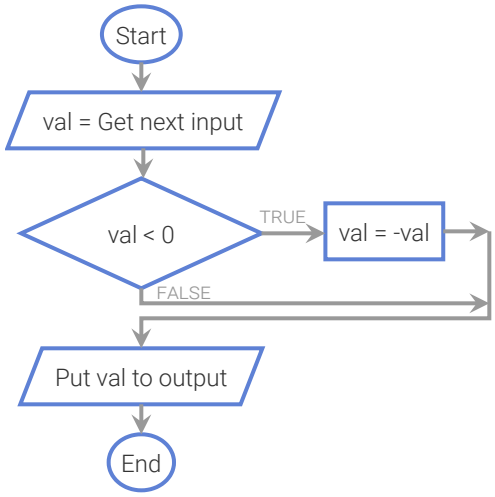
- if-else
- if-not

If branch example: Absolute value

The following shows how an if branch can be used to compute an absolute value of a number.

PARTICIPATION ACTIVITY 3.1.3: Computing absolute value.

Full screen



Variables

0 val integer

Input

-99

Output

ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 3.1.4: Example if branch: Absolute value.

Consider the example above.

1) If the input is -6, does the true branch execute?
   - Yes
   - No

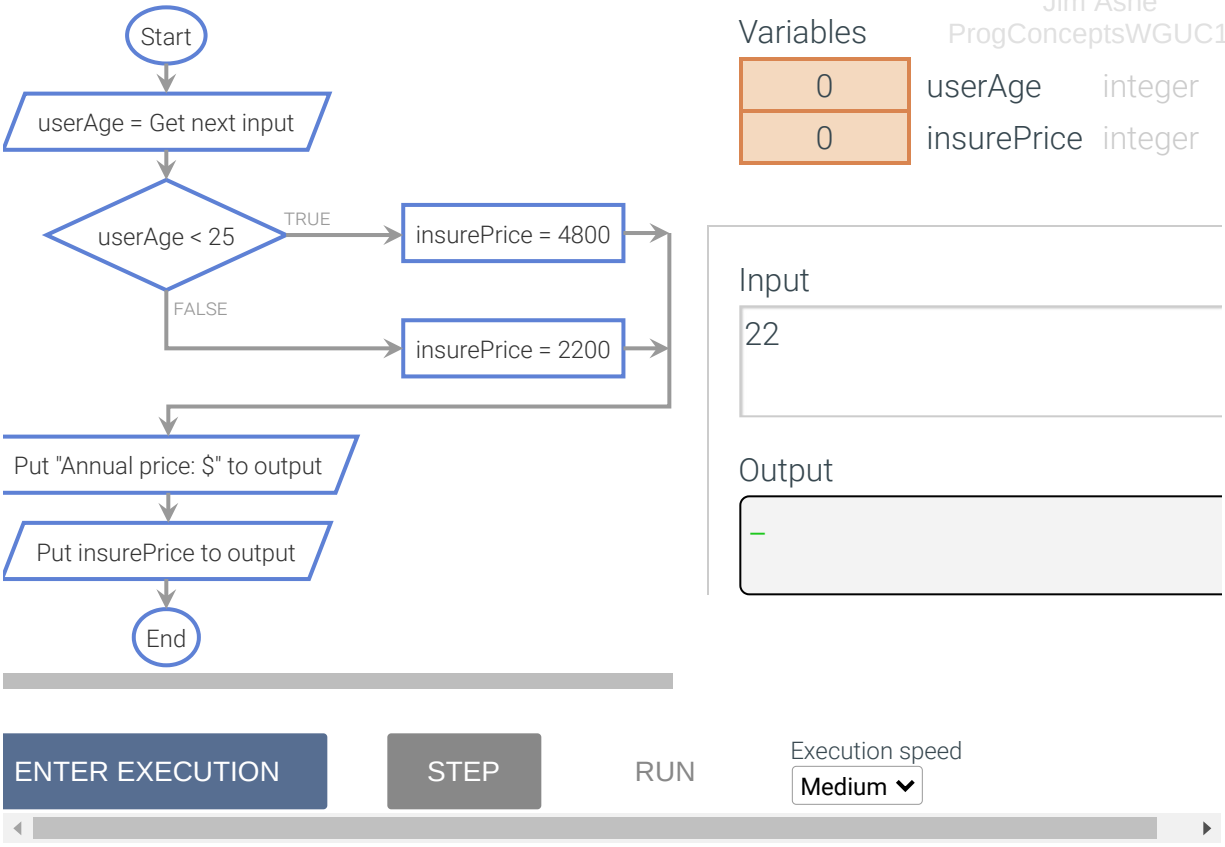
2) If the input is 2, does the true branch execute?
   - Yes
   - No

If-else branches example: Insurance prices

In the example below, if a user inputs an age less than 25, the statement `insurePrice = 4800` executes. Else, `insurePrice = 2200` executes.

PARTICIPATION ACTIVITY 3.1.5: Insurance price.

Full screen



Car insurance prices

(Car insurance prices for drivers under 25 are higher because 1 in 6 such drivers are involved in an accident each year, vs. 1 in 15 for older drivers. Source: www.census.gov, 2009).

PARTICIPATION ACTIVITY 3.1.6: Branches.

Consider the insurance price example above.

1) If userAge is 18, what price is

output?

- 4800
- 2200

2) If userAge is 30, what price is output?

- 4800
- 2200

3) If userAge is 25, what price is output?

- 4800
- 2200

4) Is there any value for userAge that executes both branches?

- Yes
- No

5) Is there any value for userAge that executes neither branch?

- Yes
- No

6) For any value of userAge, do the two output statements always execute?

- Yes
- No

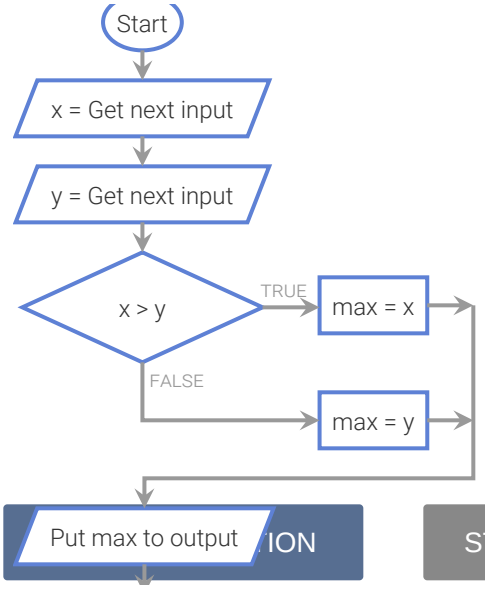
Example with if-else branches: Max

The following example shows how if-else branches can be used to get the maximum of two values.

PARTICIPATION ACTIVITY 3.1.7: If-else branches example: Max. Full screen

Variables

0	x	integer
0	y	integer
0	max	integer



Input

55 79

Output

STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 3.1.8: If-else example: Max.

Consider the example above.

1) When the input is -3 0, which branch executes?
   - If
   - Else

2) When the input is 99 98, which branch executes?
   - If
   - Else

3) The if branch assigns max = x. The else branch assigns max = ?
   - x
   - y

4) If the inputs are 5 5, does max get assigned with x or y?
   - x
   - y

If-elseif branches

A branch may itself contain a decision and branches. Commonly, a series of decisions appear cascaded in each decision's false branch, known as ***if-elseif branches*** , to detect specific values of a variable. The example below detects values of 1, 25, or 50 for variable numYears.

The example below uses ==. The ***equality operator ==*** evaluates to true if the left and right sides are equal. Ex: If numYears is 10, then `numYears == 10` evaluates to true. Note the equality operator is ==, not =.

PARTICIPATION ACTIVITY 3.1.9: If-elseif example: Anniversaries.

Animation content:

```
A zyFlowchart program follows:
integer numYears

numYears = Get next input

if numYears == 1
   Put "Newlyweds" to output
else
   if numYears == 25
      Put "Silver" to output
   else
      if numYears == 50
         Put "Golden" to output
      else
         Put "Congrats" to output

Variables in memory is as follows:
0 numYears: integer
```

Animation captions:

1. This cascaded decision structure is common to detect a specific value of a variable. If numYears is 1, the first branch executes.
2. Else, if numYears is 25, the second branch executes. Else, if numYears is 50, the third branch executes.
3. Else, the last branch executes.

PARTICIPATION ACTIVITY 3.1.10: If-elseif branches.

Consider the anniversaries example above. Type the output for the given values.

1) numYears is 25

Check Show answer

2) numYears is 1

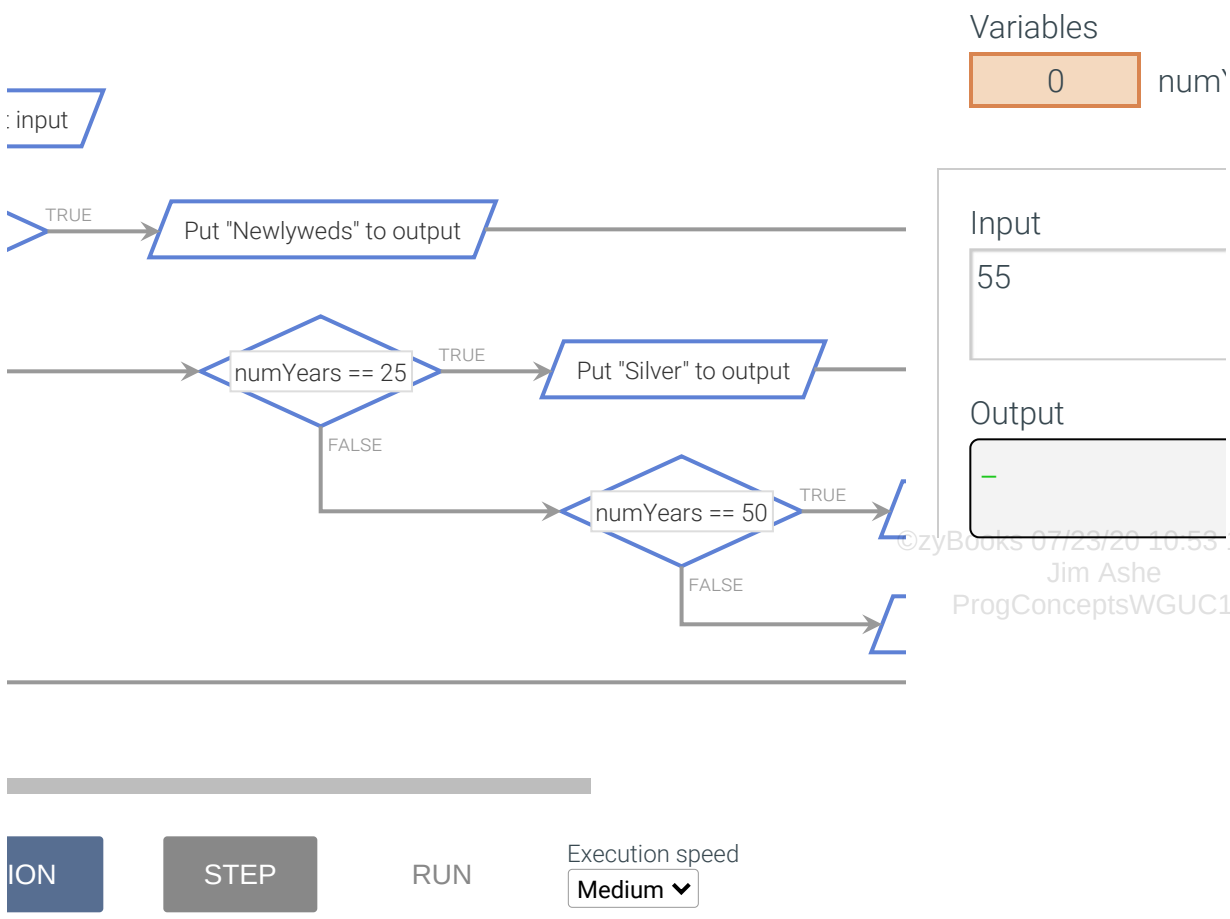
Check Show answer

3) numYears is 75

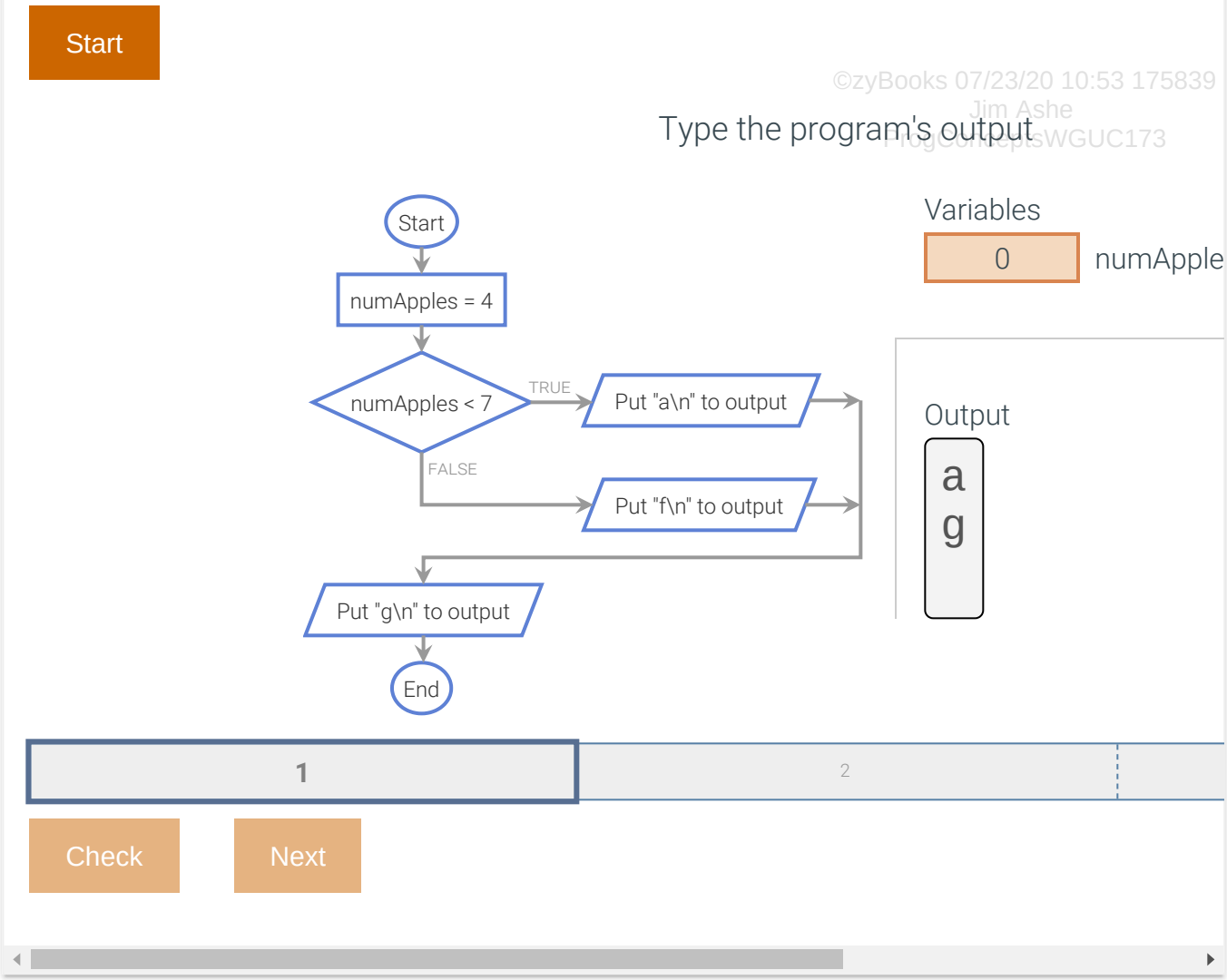
Check Show answer

For convenience, the above animation's example appears below in the zyFlowchart tool.

PARTICIPATION ACTIVITY 3.1.11: If-elseif example: Anniversaries. Full screen



CHALLENGE ACTIVITY 3.1.1: Enter the output for the if-else branches.



How was this section?

Provide feedback

3.2 More branches

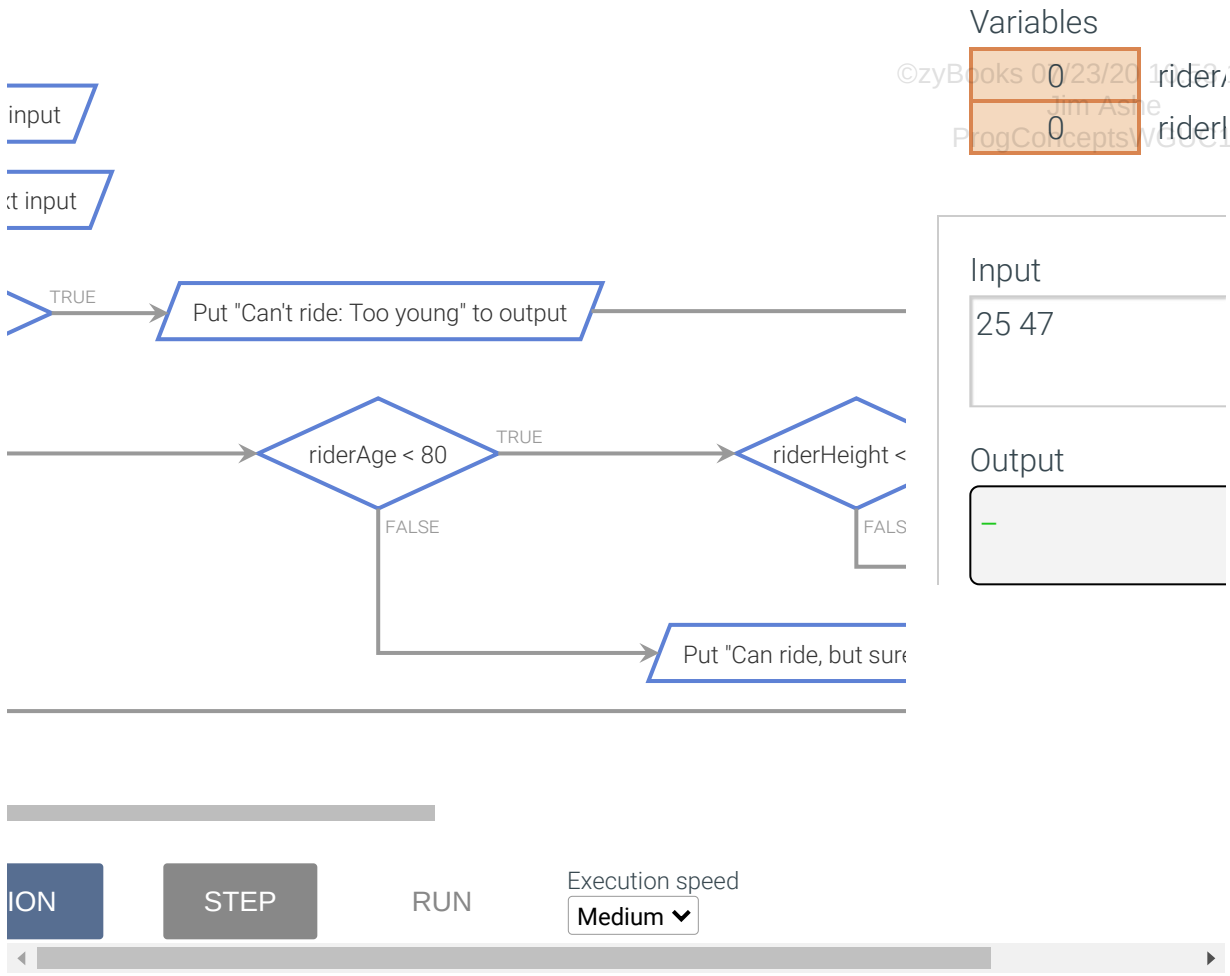
Nested if-else branches

If-else branches have two branches. A branch's statements can include any valid statements, including another if-else branch, known as ***nested branches***. The nested branches can take on

various forms, and the if-else branches may even use different variables.

PARTICIPATION ACTIVITY 3.2.1: Nested branches: Amusement park rider age and height example.

Full screen



PARTICIPATION ACTIVITY 3.2.2: Nested branches.

Consider the example above.

1) Indicate the output given input: 5 60

- Can't ride: Too young
- Can't ride: Too short
- Can ride
- Can ride, but sure?

2) Indicate the output given input: 10 60

- Can't ride: Too young

- Can't ride: Too short
- Can ride
- Can ride, but sure?

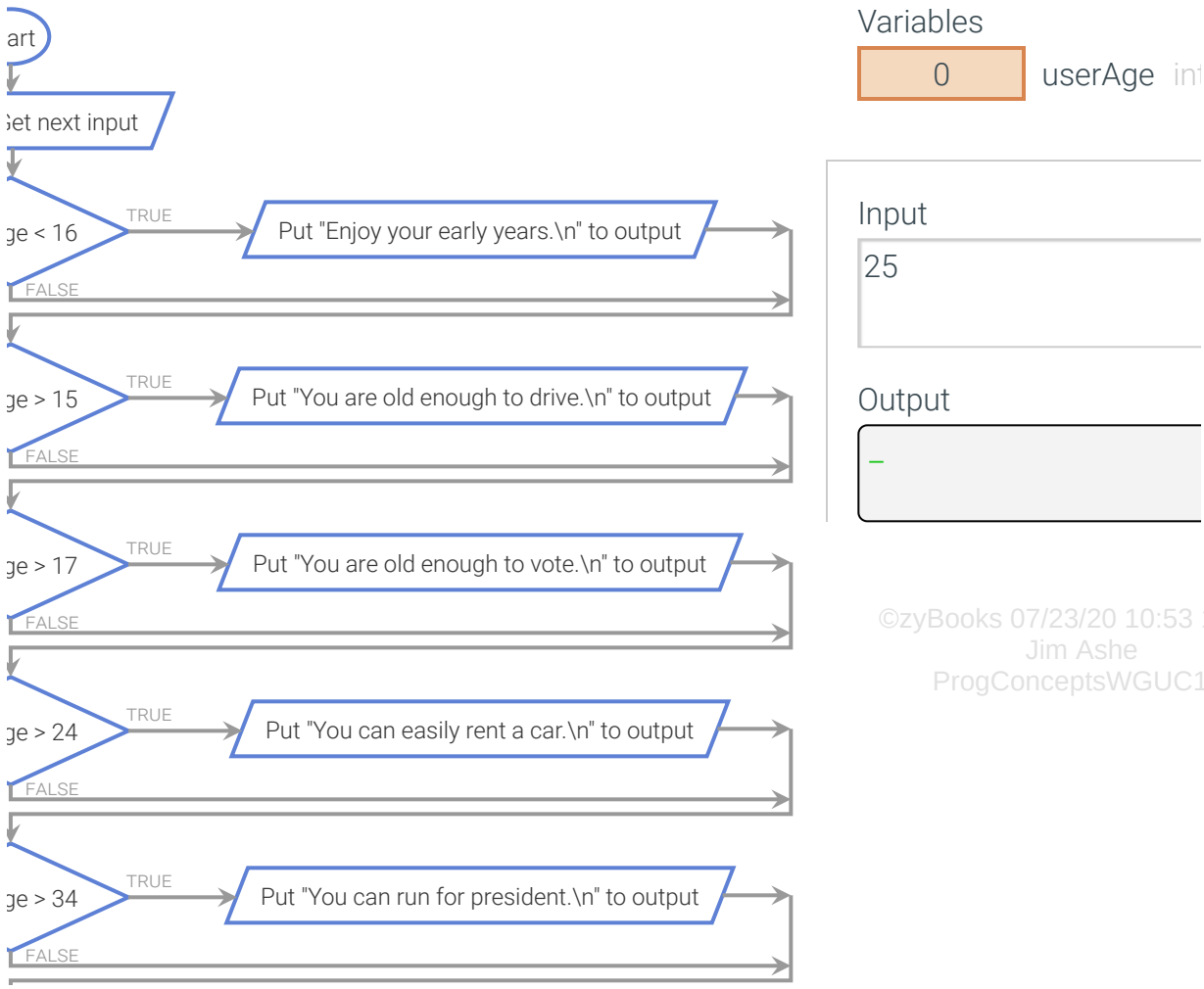
3) Which input would yield output of "Can ride, but sure?"?

- 50 60
- 85 60
- 60 85

Multiple if branches

Sometimes the programmer has multiple decisions in sequence, which look similar to an if-elseif branches but have a very different meaning. In ***multiple if branches***, each decision is independent, and thus more than one branch can execute, in contrast to the if-elseif arrangement where only one branch can execute.

PARTICIPATION ACTIVITY 3.2.3: Multiple distinct decisions. Full screen



XECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 3.2.4: Multiple distinct if branches.

Consider the example above. Match the input with the produced output.

No such input **40** **21** **14**

	Enjoy your early years.
	You are old enough to drive. You are old enough to vote.
	You are old enough to drive. You are old enough to vote. You can easily rent a car. You can run for president.
	Enjoy your early years. You are old enough to drive.

Reset

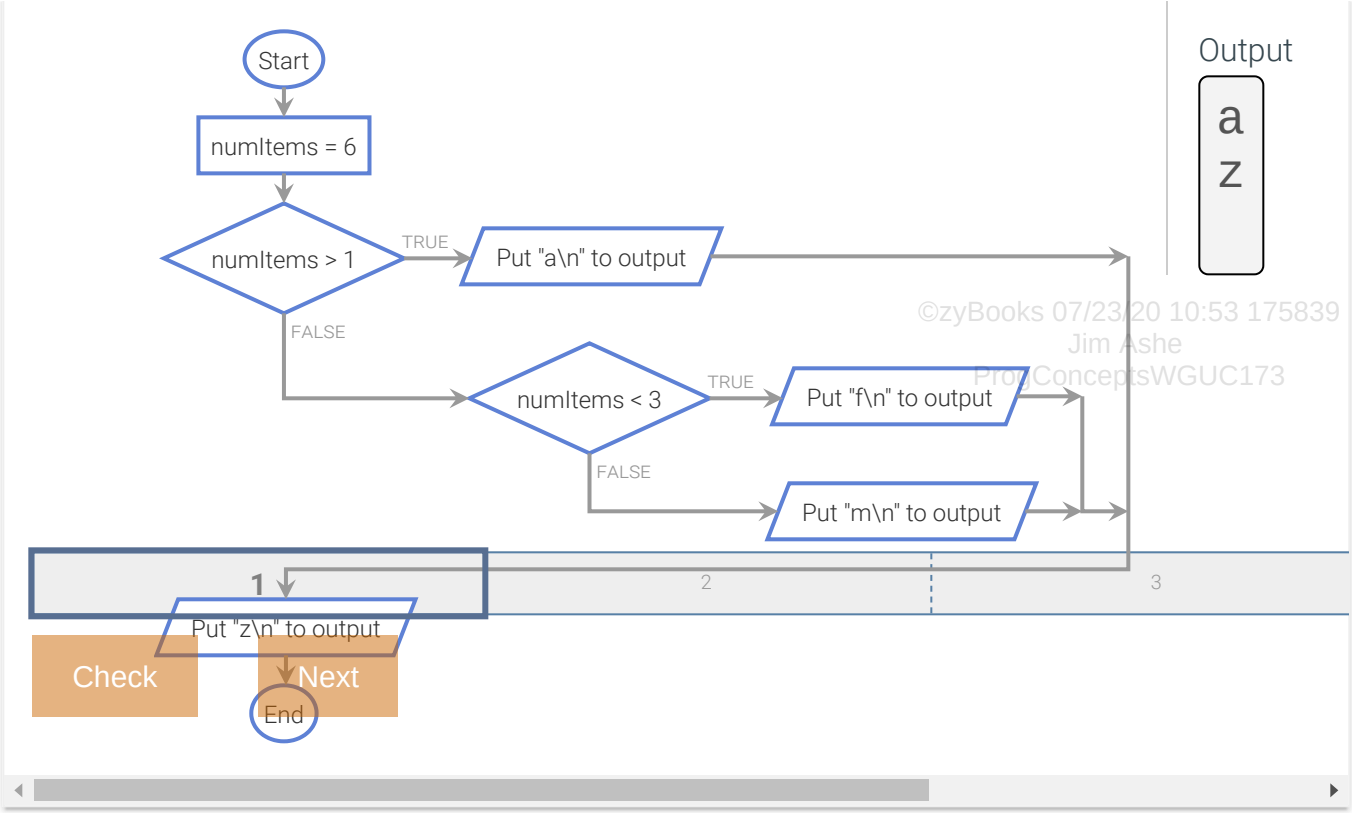
CHALLENGE ACTIVITY 3.2.1: Enter the output for the multiple if-else branches.

Start

Type the program's output

Variables

0



How was this section?

Provide feedback

3.3 Equality and relational operators

Equality operators

An ***equality operator*** checks whether two operands' values are the same (==) or different (!=). Note that equality is ==, not just =.

An expression involving an equality operator evaluates to a Boolean value. A ***Boolean*** is a type that has just two values: true or false.

Table 3.3.1: Equality operators.

Equality operators	Description	Example (assume x is 3)
==	a **==** b means a is equal to b	x == 3 is true x == 4 is false

!=	a **!=** b means a is not equal to b	x != 3 is false x != 4 is true

PARTICIPATION ACTIVITY 3.3.1: Evaluating expressions that have equality operators.

Indicate whether the expression evaluates to true or false.
x is 5, y is 7.

1) x == 5

- True
- False

2) x == y

- True
- False

3) y != 7

- True
- False

4) y != 99

- True
- False

5) x != y

- True
- False

PARTICIPATION ACTIVITY 3.3.2: Creating expressions with equality operators.

Type the operator to complete the desired expression.

1) numDogs is 0

```
numDogs  0
```

Check Show answer

2) numDogs and numCats are the same

```
numDogs  numCats
```

Check Show answer

3) numDogs and numCats differ

```
numDogs  numCats
```

Check Show answer

4) numDogs is either less-than or greater-than numCats

```
numDogs  numCats
```

Check Show answer

Relational operators

A ***relational operator*** checks how one operand's value relates to another, like being greater than.

Some operators like >= involve two characters. A programmer cannot arbitrarily combine the >, =, and < symbols; only the shown two-character sequences represent valid operators in the zyFlowchart language (and most languages).

Table 3.3.2: Relational operators.

Relational operators	Description	Example (assume x is 3)
<	a **<** b means a is less-than b	x < 4 is true x < 3 is false
>	a **>** b means a is greater-than b	x > 2 is true x > 3 is false
<=	a **<=** b means a is less-than-or-equal to b	x <= 4 is true x <= 3 is true x <= 2 is false
	a **>=** b means a is greater-than-or-equal to	x >= 2 is true

>=	b	x >= 3 is true x >= 4 is false

PARTICIPATION ACTIVITY 3.3.3: Evaluating equations having relational operators.

Indicate whether the expression evaluates to true or false.
x is 5, y is 7.

1) x <= 7

- True
- False

2) y >= 7

- True
- False

3) Is x <> y a valid expression?

- Yes
- No

4) Is x =< y a valid expression?

- Yes
- No

PARTICIPATION ACTIVITY 3.3.4: Creating expressions with relational operators.

Type the operator to complete the desired expression.

1) numDogs is greater than 10

```
numDogs  10
```

Check Show answer

2) numCars is greater than or equal to 5

```
numCars  5
```

Check Show answer

3) numCars is 5 or greater

`numCars` [ ] 5

Check Show answer

4) centsLost is a negative number

`centsLost` [ ] 0

Check Show answer

Example: Golf scores

In golf and miniature golf, each hole has a "par" indicating the normal number of strokes to get the ball in the hole. The following program outputs special names for numbers below par. The program uses one relational operator, and several equality operators, in a multi-branch if-else statement.

PARTICIPATION ACTIVITY 3.3.5: Golf scores. Full screen

This program outputs special names for different numbers of strokes for a par 4 hole. Press Run to see the name for the input strokes of 2.

Variables

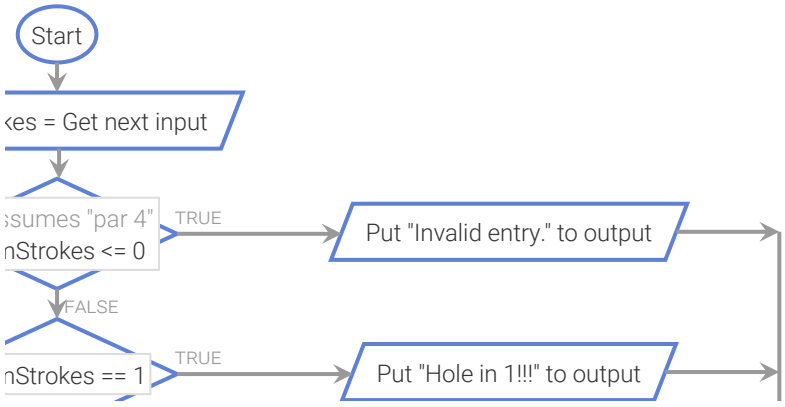
0 numStrokes

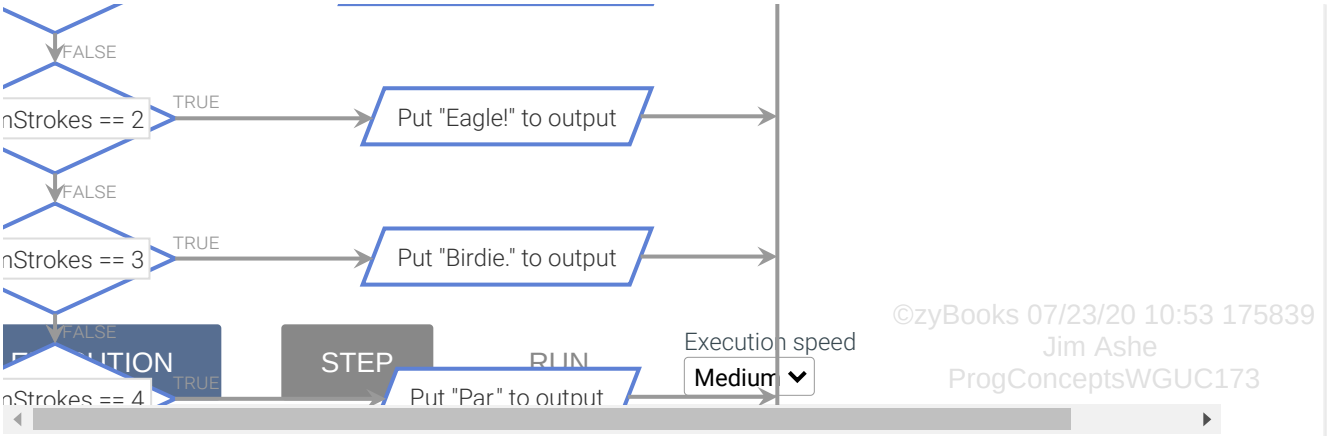
Input

2

Output

–





Comparing floating-point types

The relational and equality operators work for integer and floating-point built-in types.

However, floating-point types should *not* be compared using the equality operator ==, due to the imprecise representation of floating-point numbers, as discussed in a later section.

PARTICIPATION ACTIVITY 3.3.6: Comparing various types.

Which comparison will compile AND consistently yield expected results? Variables have types denoted by their names.

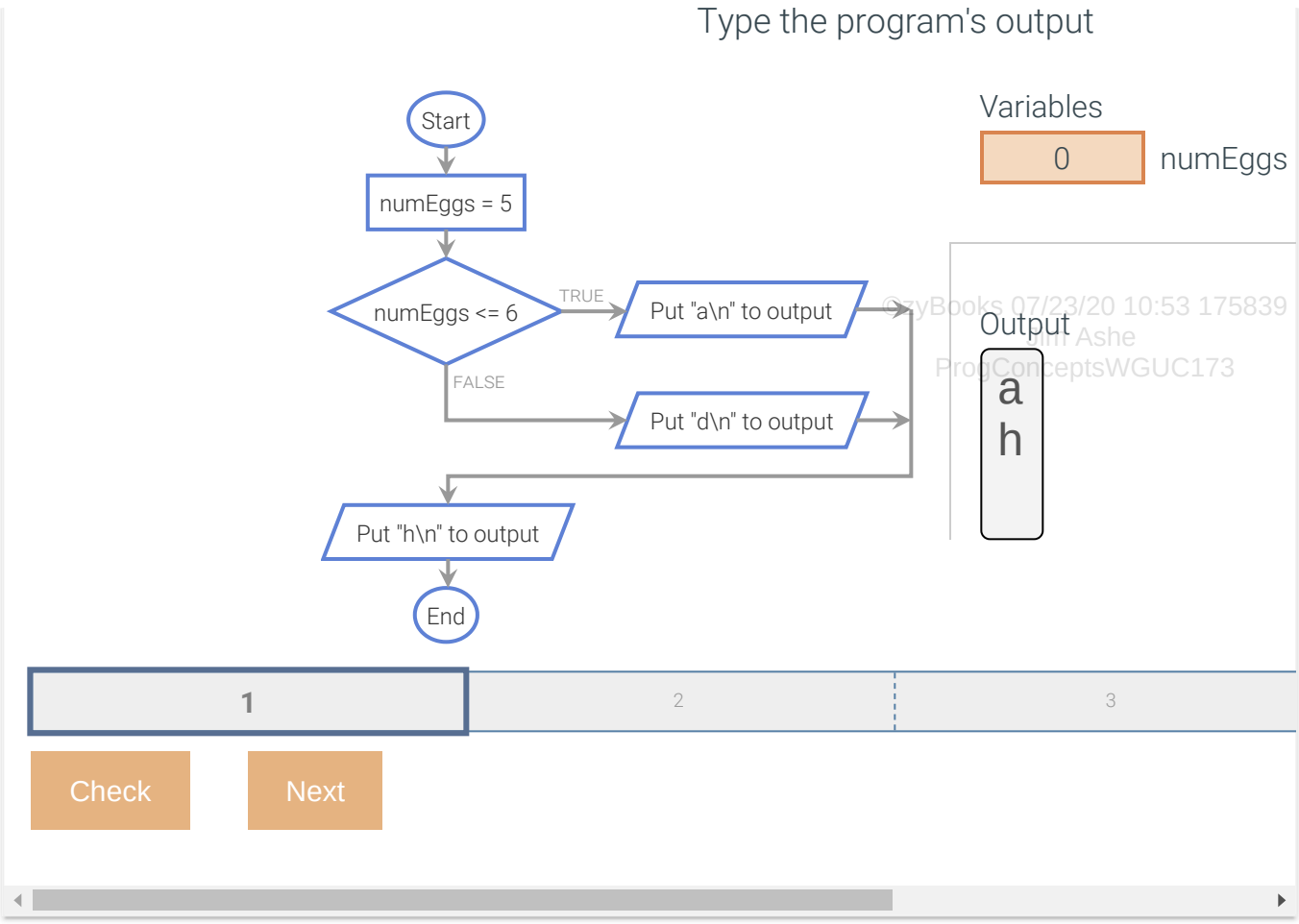
1) myInt == 42
   - OK
   - Not OK

2) myFloat == 3.25
   - OK
   - Not OK

3) myInt == myFloat
   - OK
   - Not OK

CHALLENGE ACTIVITY 3.3.1: Enter the output for the branches with relational and equality operators.

Start



How was this section?

Provide feedback

3.4 Detecting ranges using branches

Decision sequences and ranges

People regularly deduce ranges based on a decision sequence. Ex: In a soccer league, no teams may exist for players 5 and under. So, a "U8" division ("Under 8") is deduced to be for ages 6-7, and not for 1-7 because players 5 and under are already excluded. Likewise, U10 would be 8-9 because 6-7 is already covered by U8.

PARTICIPATION ACTIVITY 3.4.1: Deducing ranges by a decision sequence.

Animation content:

```
If age < 6: No teams
Else If age < 8: Play on U8 team
Else If age < 10: Play on U10 team
Else If age < 12: Play on U12 team
Else: No teams
```

Animation captions:

1. Kids of various ages may wish to play soccer. A soccer club may not have teams for kids 5 and under.
2. One level of teams is listed as "Under 8" (or U8), which is understood to mean just for ages 7 and 6, but not 5 and younger.
3. Likewise, U10 means ages 9 and 8, and U12 means 11 and 10. No teams exist for ages 12 and over.
4. Using a sequence of decisions enables a concise way of specifying a range, such as saying U12 rather than saying ages 10 and 11.

PARTICIPATION ACTIVITY 3.4.2: Using a decision sequence to detect increasing ranges.

Indicate the range covered by each decision in the following decision sequence, meaning a decision is considered only if the previous decision was false. x is a non-negative integer.

30+ **20 - 29** **10 - 19** **0 - 9**

	x < 10
	x < 20
	x < 30
	Else

Reset

PARTICIPATION ACTIVITY 3.4.3: More ranges with a decision sequence.

Indicate the range detected by the expression, assuming each question continues a decision sequence, with the next decision being reached only if the previous decision was false. x is an integer. Type ranges as: 25 - 29

1) x > 100

- infinity

Check Show answer

2) x > 50

Check Show answer

3) Else

-infinity -

Check Show answer

4) Is this a reasonable decision sequence? Type yes or no.

x < 100: Do A
x < 200: Do B
x < 150: Do C
Else: Do D

Check Show answer

Using cascaded decisions to detect ranges

Programmers commonly use decisions cascaded in an if-elseif structure to detect ranges of numbers. In the following example, the second decision is only considered if the first decision is false. So the second branch is executed if userAge < 16 *is false* (so 16 or greater) AND userAge is < 25, meaning userAge is between 16 - 24 (inclusive).

PARTICIPATION ACTIVITY 3.4.4: Using sequential nature of multi-branch if-else for ranges: Insurance prices.

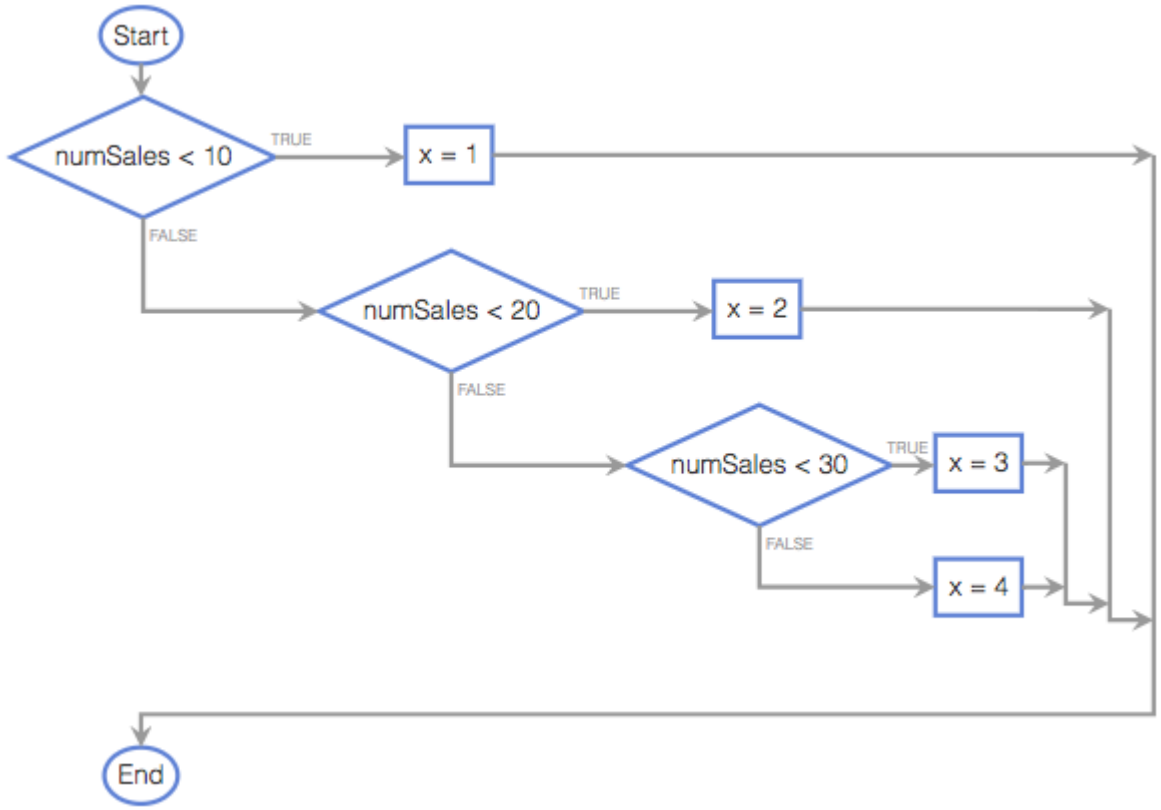
This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.4.5: Decision sequences and ranges.

Type the range for each branch. Type ranges as 25-29, or as 30+ for 30 and up.



1) Range for x = 2

Check Show answer

2) Range for x = 3

Check Show answer

3) Range for x = 4

Check Show answer

How was this section?

Provide feedback

3.5 Logical operators

Logical and, or, not

A ***logical operator*** treats operands as being true or false, and evaluates to true or false. Logical operators include and, or, not.

PARTICIPATION ACTIVITY 3.5.1: Logical operators: and, or, not.

Animation content:

```
First table:
a is false, b is false, a and b yields false
a is false, b is true, a and b yields false
a is true, b is false, a and b yields false
a is true, b is true, a and b yields true

Let x = 7, y = 9
(x > 0) and (y < 10) evaluates to true and true, which is true
(x > 0) and (y < 5) evaluates to true and false, which is false

Second table:
a is false, b is false, a and b yields false
a is false, b is true, a and b yields true
a is true, b is false, a and b yields true
a is true, b is true, a and b yields true

Let x = 7, y = 9
```

```
(x > 0) or (y > 10) evaluates to false or false, which is false
(x < 0) or (y > 5) evaluates to false or true, which is true

Third table:
a is false, not a yields true
a is true, not a yields false

Let x = 7
not (x < 0) evaluates to true
not (x > 0) evaluates to false
```


Animation captions:

1. The and operator evaluates to true only if BOTH operands are true. Each operand (a, b above) is an expression that evaluates to true or false.
2. The or operator evaluates to true if ANY operand is true (a, b, or both).
3. The not operator evaluates to the opposite of the operand.
4. Each operand is an expression. If x = 7, y = 9, then (x > 0) and (y < 10) is true and true, so the and operator evaluates to true (both operands are true).

Table 3.5.1: Logical operators.

Logical operator	Description
(a) and (b)	***Logical and***: true when both operands are true
(a) or (b)	***Logical or***: true when at least one of the two operands is true
not(a)	***Logical not***: true when the one operand is false, and vice-versa

PARTICIPATION ACTIVITY 3.5.2: Evaluating expressions with logical operators.

Indicate whether the expression evaluates to true or false.
x is 7, y is 9.

1) x > 5

- ○ true
- ○ false

2)

(x > 5) and (y < 20)

- true
- false

3) (x > 10) and (y < 20)

- true
- false

4) (x > 10) or (y < 20)

- true
- false

5) (x > 10) or (y > 20)

- true
- false

6) not(x > 10)

- true
- false

7) not((x > 5) and (y < 20))

- true
- false

Detecting ranges with logical operators (general)

A common use of logical operators is to detect if a value is within a range.

PARTICIPATION ACTIVITY 3.5.3: Using AND to detect if a value is within a range.

Animation captions:

1. In mathematics, the range 10 < x < 15 means that x may be 11, 12, 13, 14.
2. In a program, such a range is specified using two < operators along with the and operator. 10 < x defines the range above 10.
3. x < 15 defines the range below 15. Adding the and operator yields the overlapping range. Only when x is 11, 12, 13, or 14 will both expressions be true.

PARTICIPATION ACTIVITY 3.5.4: Using the and operator to detect if a value is within a range.

1) Which approach uses a logical operator to detect if x is in the range 1 to 99.
   - 0 < x < 100
   - (0 < x) and (x < 100)
   - (0 < x) and (x > 100)

2) Which detects if x is in the range -4 to +4?
   - (x < -5) and (x < 5)
   - (x > -5) or (x < 5)
   - (x > -5) and (x < 5)

3) Which detects if x either less than -5 or greater than 10?
   - (x < -5) and (x > 10)
   - (x < -5) or (x > 10)

Evaluating expressions

Expressions within parentheses are evaluated first to yield true or false, then higher-level expressions are evaluated.

PARTICIPATION ACTIVITY 3.5.5: Evaluating expressions with logical operators.

Given numPeople = 10, userVal = 55. Indicate whether the expression evaluates to true or false.

1) `(numPeople >= 10) and (userVal == '2')`
   - true
   - false

2) `(numPeople >= 20) or (userVal == 55)`
   - true
   - false

3) `not(userVal == 29)`

- true
- false

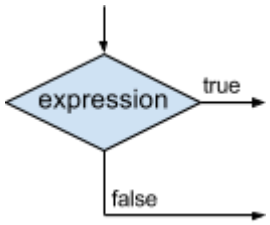
4) `not( (numPeople == 5) or (numPeople == 6) )`

- true
- false

Logical operators in decision expressions

Logical operators are commonly used in expressions found in decision expressions.

PARTICIPATION ACTIVITY | 3.5.6: Logical operators: Complete the decision expressions to detect the desired range.



1) daysLogged is greater than 30 and less than 90

```
(daysLogged > 30)
(daysLogged < 90)
```

Check Show answer

2) 0 < maxCars < 100

```
(maxCars > 0)
(maxCars < 100)
```

Check Show answer

3) numStores is between 10 and 20, inclusive.

```
(numStores >= 10)
(numStores <= 20)
```

Check Show answer

4) userAge is either less than 15, or greater than 79.

```
(userAge < 15)
(userAge > 79)
```

Check Show answer

PARTICIPATION ACTIVITY 3.5.7: Creating expressions with logical operators.

1) numDogs is 3 or more and numCats is 3 or more.

```
(numDogs >= 3)
```

Check Show answer

2) wage is greater than 10 or age is less than 18. Use or. Use >, < (rather than >=, <=).

Check Show answer

3) num is a 3-digit positive integer. Ex: 100, 989, and 523 are each a 3-digit positive integer, but 55, 1000, and -4 are not.

For most direct readability, your expression should compare directly with the smallest and largest 3-digit number.

```
(num >= 100)
```

Check Show answer

Example: TV channels

A cable TV provider may have regular channels numbered 2-499, and high-definition channels numbered 1002-1499. A program may set a character variable to 's' for standard, 'h' for high-definition, and 'e' for error.

PARTICIPATION ACTIVITY 3.5.8: Detecting ranges: Cable TV channels.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.5.9: TV channel example: Detecting ranges.

Consider the example above.

1) If userChannel is 300, what does the first expression, `(userChannel >= 2) and (userChannel <= 499)` evaluate to?
   - ○ true
   - ○ false

2) If userChannel is 300, does the second expression get checked?
   - ○ Yes
   - ○ No

3) Did the expressions use logical and or logical or?
   - ○ and
   - ○ or

4) Channels 500-599 are pay channels. Does this expression detect that range? `(userChannel >= 500) or (userChannel <= 599)`

- Yes
- No

Detecting ranges implicitly vs. explicitly

If a program should detect increasing ranges without gaps, cascaded decisions can be used without logical operators; the low-end of the range is implicitly known upon reaching an expression. Likewise, a decreasing range without gaps has implicitly-known high-ends. In contrast, when gaps exist, the range's low and high ends must both be explicitly detected, using a logical operator.

PARTICIPATION ACTIVITY 3.5.10: Detecting ranges explicitly (left) vs. implicity (right).

Animation content:

```
A zyFlowchart snippet appears on the left:
if userNum < 0
   // value is less than 0
else
   if (userNum >= 0) and (userNum < 10)
      // value is between 0 to 9
   else
      if (userNum >= 10) and (userNum < 20)
         // value is between 10 and 10

A zyFlowchart snippet appears on the right:
if userNum < 0
   // value is less than 0
else
   if (userNum < 10)
      // userNum must be >= 0 for first if statement to evaluate to
false
      // value is between 0 to 9
   else
      if (userNum < 20)
         // userNum must be >= 10 for first and second if statement
to evaluate to false
```

```
            // value is between 10 and 20
```

Animation captions:

1. A common cascaded arrangement of decisions detects ranges of a variable. The first true branch executes for values < 0, the second for 0-9, and the third for 10-19.
2. If the first decision's userNum < 0 yielded false, userNum must be >= 0, so the next decision need not check for >= 0, and can just check for userNum < 10.
3. Likewise for the next decision: userNum must be >= 10, so only < 20 is checked. Such implicit ranges have easier-to-read expressions.

PARTICIPATION ACTIVITY 3.5.11: Detecting ranges implicitly vs. explicitly.

Using cascaded decisions, indicate whether the ranges can be detected implicitly, or must be detected explicitly.

1) <= 10
11..20
21..50
51+

- Implicit
- Explicit

2) <=10
50..100
150..200

- Implicit
- Explicit

3) >= 100
90..99
80..89

- Implicit
- Explicit

How was this section?

Provide feedback

3.6 Order of evaluation

Precedence rules

The order in which operators are evaluated in an expression are known as ***precedence rules***. Arithmetic, logical, and relational operators are evaluated in the order shown below.

Table 3.6.1: Precedence rules for arithmetic, logical, and relational operators.

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In `(a * (b + c)) - d`, the + is evaluated first, then *, then -.
not	Logical not is next	`not(x == 1) or y` is evaluated as `(not(x == 1)) or y`
* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	`z - 45 * y < 53` evaluates * first, then -, then <.
`< <= > >=`	Relational operators	`x < 2 or x >= 10` is evaluated as `(x < 2) or (x >= 10)` because < and >= have precedence over the or operator.
`== !=`	Equality and inequality operators	`x == 0 and x >= 10` is evaluated as `(x == 0) and (x >= 10)` because == and >= have precedence over the and operator.
and	Logical and	`x == 5 or y == 10 and z != 10` is evaluated as `(x == 5) or ((y == 10) and (z != 10))` because the and operator has precedence over the

		or operator.
or	Logical OR	or has the lowest precedence of the listed arithmetic, logical, and relational operators.

PARTICIPATION ACTIVITY 3.6.1: Applying the precedence rules to an expression can be thought of as a 'tree'.

Animation captions:

1. Expressions like x + 1 > y * z or z == 3 are evaluated using precedence rules. Among +, >, *, or, ==, the * comes first.
2. Next comes +, then ==, and finally or.
3. The expression is actually treated like a "tree", evaluated from the bottom upwards.
4. If x is 7, y is 6, and z is 3, then y * z is 18. Next, x + 1 is 8. Next, 8 > 18 is false. Next, z == 3 is true. Finally, false or true is true.

PARTICIPATION ACTIVITY 3.6.2: Order of evaluation.

To teach precedence rules, these questions intentionally omit parentheses; good style would use parentheses to make order of evaluation explicit.

1) After y == 1 is evaluated, which operator is evaluated next?
not(y == 1) and x == 2

- ○ and
- ○ not

2) Which operator is evaluated first?
w + 3 > x - y * z

- ○ +
- ○ -
- ○ >
- ○ *

3) In what order are the operators evaluated?
w + 3 != y - 1 and x == 1

- ○ +, !=, -, and, ==

- +, -, and, !=, ==
- +, -, !=, ==, and

4) What does this expression evaluate to, given x = 4, y = 7.
`x == 3 or x + 1 > y`

- true
- false

Common error: Missing parentheses

A common error is to write an expression that is evaluated in a different order than expected. Good practice is to use parentheses in expressions to make the intended order of evaluation explicit. Several examples are below.

PARTICIPATION ACTIVITY 3.6.3: Order of evaluation.

Which illustrates the actual order of evaluation via parentheses?

1) `bats < birds or birds < insects`

- ((bats < birds) or birds) < insects
- bats < (birds or birds) < insects
- (bats < birds) or (birds < insects)

2) `(num1 == 9) or (num2 == 0) and (num3 == 0)`

- (num1 == 9) or ((num2 == 0) and (num3 == 0))
- ((num1 == 9) or (num2 == 0)) and (num3 == 0)
- (num1 == 9) or (num2 == (0 and num3) == 0)

Common error: Math expression for range

A common error often made by new programmers is to write expressions like `(16 < age < 25)`, as one might see in mathematics.

PARTICIPATION ACTIVITY 3.6.4: Expression for detecting a range.

1) A programmer erroneously wrote an expression as: 0 < x < 10. Rewrite the expression using logical and. Use parentheses.

```
(0 < x)
```

Check Show answer

How was this section?

Provide feedback

3.7 Example: Toll calculation

Calculating toll based on time of day

This section presents an example program that calculates the toll amount for travel along a toll road or toll lane. The toll amount is based on the time of day, day of the week, and number of persons in the vehicle.

The initial version of the program calculates the toll amount for travel on a weekday based upon the toll schedule below. The table lists times in both am/pm format and 24-hour format.

Table 3.7.1: Weekday toll schedule.

Time (am/pm)	Time (24 hour)	Toll amount
Before 6:00 am	Before 6:00	1.55
6:00 am to 9:59 am	6:00 to 9:59	4.65
10:00 am to 5:59 pm	10:00 to 17:59	2.35

6:00 pm and after	18:00 and after	1.55

The program gets the time of travel from the user using 24 hours format, and uses the hour to determine the toll amount. If-elseif branches determine in which range the hour belongs and assign tollAmount with the toll based on the table above. Then the toll amount is output.

PARTICIPATION ACTIVITY 3.7.1: Calculating toll amount.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.7.2: Toll calculation.

For the given input, what is the final value of tollAmount?

1) 5 45

- 0.00
- 1.55
- 2.35

2) 9 45

- 1.55
- 2.35
- 4.65

3) 10 00

- 1.55
- 2.35
- 4.65

4) 22 15

- 1.55
- 2.35

Calculating toll with carpool discount

A toll road may have a discount for carpools, sometimes called high-occupancy vehicles (HOV). The following program uses if-elseif branches to adjust the toll amount based on the number of persons in the vehicle. The carpool discount rules are:

- A carpool is 3 or more persons per vehicle.
- The toll for carpools between 6:00 am and 10:00 am is half the normal toll.
- Otherwise, the toll for carpools is 0 (as in free).

PARTICIPATION ACTIVITY 3.7.3: Calculating toll amount, with carpool discount.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.7.4: Toll with carpool discount.

Match the final value of tollAmount to the timeHour and numPersons. Trace the zyFlowchart program to determine the tollAmount.

1.55 **2.325** **4.65**

	timeHour is 7, numPersons is 1
	timeHour is 8, numPersons is 4

timeHour is 20, numPersons is 2

Reset

How was this section?

Provide feedback

3.8 Floating-point comparison

This section has been set as optional by your instructor.

Floating-point numbers should not be compared using the == operator. Ex: Avoid float1 == float2. Reason: Some floating-point numbers cannot be exactly represented in the limited available memory bits like 64 bits. Floating-point numbers expected to be equal may be close but not exactly equal.

PARTICIPATION ACTIVITY 3.8.1: Floating-point comparisons.

Animation content:

```
Code snippet:
numMeters = 0.7;                // 0.7 is actually
0.69999999999999995559107901
numMeters = numMeters - 0.4;    // 0.4 is actually
0.40000000000000002220446049
numMeters = numMeters - 0.3;    // 0.3 is actually
0.29999999999999998889776975

Comments:
numMeters is expected to be 0, but is actually
-0.00000000000000005551115123
so numMeters == 0.0 evaluates to false
thus, AbsoluteValue(numMeters - 0.0) < 0.0001 should be utilized,
which will evaluate to true
```


Animation captions:

1. Floating-point numbers can't always be exactly represented in limited memory bits. 0.7 is actually 0.6999... 0.4 is 0.400...00222... 0.3 is 0.2999... Thus, 0.7 - 0.4 - 0.3 should be 0 but is actually -0.00...0055...
2. Thus, floats should not be compared with ==. If numMeters = 0.7, and then 0.4 is subtracted, then 0.3 is subtracted, then numMeters == 0.0 should yield true but will actually yield false.
3. Instead, floats should be compared for "close enough": If the difference is less than say 0.0001, the values are considered equal. Absolute value is applied since the difference could be negative.

Floating-point numbers should be compared for "close enough" rather than exact equality. Ex: If (x - y) < 0.0001, x and y may be deemed equal. Because the difference may be negative, the absolute value is used: AbsoluteValue(x - y) < 0.0001. AbsoluteValue() is a math function built into zyFlowchart. The difference threshold indicating that floating-point numbers are equal is often called the ***epsilon***. Epsilon's value depends on the program's expected values, but 0.0001 is common.

PARTICIPATION ACTIVITY 3.8.2: Using == with floating-point numbers.

1) Given: float x, y
x == y is OK.
   - True
   - False

2) Given: float x
x == 32.0 is OK.
   - True
   - False

3) Given: integers x and y
x == y is OK.
   - True
   - False

4) Given: float x
x == 32 is OK.
   - True
   - False

PARTICIPATION ACTIVITY 3.8.3: Floating-point comparisons.

Each comparison has a problem. Click on the problem.

1) AbsoluteValue (x - y) == 0.0001

2) AbsoluteValue (x - y) < 1.0

PARTICIPATION ACTIVITY 3.8.4: Floating-point comparisons.

Complete the comparison for floating-point numbers.

1) Determine if float variable x is 98.6.

```
          (x - 98.6)
< 0.0001
```

Check Show answer

2) Determine if float variables x and y are equal. Threshold is 0.0001.

```
AbsoluteValue(x - y)
```

Check Show answer

3) Determine if float variable x is 1.0

```
AbsoluteValue(          )
< 0.0001
```

Check Show answer

Example: Body temperature

The example below shows how floating-point values should be compared in a program, comparing for close enough, rather than using the == operator.

PARTICIPATION ACTIVITY 3.8.5: Example: Body temperature.

This activity failed to load. Please

try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.8.6: Body temperature in Fahrenheit.

Refer to the body temperature example above.

1) What is output if the user enters 98.6?
   - Exactly normal
   - Above normal
   - Below normal

2) What is output if the user enters 97.0?
   - Exactly normal
   - Above normal
   - Below normal

3) What is output if the user enters 98.6000001?
   - Exactly normal
   - Above normal
   - Below normal

Observing inexact representation of floating-point values

The tool below allows a user to type a value and see how that value is actually represented in a 32-bit floating-point representation. The user can ignore the 0's and 1's, and just notice the actual represented value. One might type 0.7 and notice the represented value is 0.6999... Other numbers like 0.25 can be represented exactly.

PARTICIPATION ACTIVITY 3.8.7: Inexact representation of floating-point values.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.8.8: Representing floating-point numbers.

1) Floating-point values are always stored with some inaccuracy.
   - True
   - False

2) If a floating-point variable is assigned with 0.2, and putting that variable to output yields 0.2, the value must have been represented exactly.
   - True
   - False

How was this section?

Provide feedback

3.9 Pseudocode: Branches

If statements and if-else statements

In pseudocode, if-else branches are written as an ***if-else statement***: An if expression with the true branch's sub-statements, followed by an else part with any false branch sub-statements. An ***if statement*** is an if expression followed by sub-statements, with no else part.

PARTICIPATION

ACTIVITY 3.9.1: Pseudocode for if and if-else statements.

Animation content:

The following program is shown, graphically on the left, and textually on the right.


```
integer hotelRate
integer userAge

hotelRate = 155
userAge = Get next input
if userAge > 65
   hotelRate = hotelRate - 20
Put "Your rate: " to output
Put hotelRate to output
```

Animation captions:

1. In pseudocode, a decision uses the word "if" followed by the expression.
2. The true branch's sub-statements start on the next line and are indented.
3. If the expression is true, the true branch executes. If the expression is false, execution just proceeds to the subsequent statements.
4. For a program having statements in the else (false) branch, the pseudocode's if-else statement has an else part, with indented sub-statements as well.

PARTICIPATION ACTIVITY 3.9.2: If-else statement in pseudocode.

For the given if-else statement, select the final value of insurePrice for the indicated value of userAge.

```
if userAge < 25
   insurePrice = 4800
else
   insurePrice = 2200
```


1) userAge = 20

- 2200
- 4800

2) userAge = 30

- 2200
- 4800

3) userAge = 25

- 2200
- 4800

4) Can a particular value of userAge cause both the if and else branches to execute?

- Yes
- No

5) Can a particular value of userAge cause neither of the if or else branches to execute?

- Yes
- No

If-elseif statements

An ***if-elseif statement*** starts with an if expression, followed by elseif expressions, and ending with else; when a program reaches the statement, exactly one of those branches will execute. When the else branch has no statements, the else part is omitted.

PARTICIPATION ACTIVITY 3.9.3: if-elseif statement.

Animation captions:

1. In an if-elseif statement, the first expression that evaluates to true has its branch execute.
2. Execution then jumps to after the if-elseif statement.

PARTICIPATION ACTIVITY 3.9.4: If-else branches in pseudocode.

Consider the example above. What is the program's output for each value of numYears?

1) numYears = 1

Check Show answer

2) numYears = 50

Check Show answer

3) numYears = 5

Check Show answer

4) numYears = -1

Check Show answer

5) Is there any value of numYears that will cause two of the if-elseif branches to execute? Type yes or no.

Check Show answer

6) Is there any value of numYears that will cause none of the if-elseif branches to execute? Type yes or no.

Check Show answer

Multiple if statements

Multiple distinct if statements are simply written consecutively. Each if statement is considered independently of the others, so multiple branches may execute. In contrast, an if-elseif statement is one large statement of which only one branch can execute.

PARTICIPATION ACTIVITY 3.9.5: Multiple if statements in pseudocode.

Animation captions:

1. Programmers often use multiple if statements. While looking similar to an if-elseif statement, the if statements are independent of each other.
2. If userAge is 18, the first if's expression is true so that branch executes. The second if's expression is true so that branch executes. But not the third.

PARTICIPATION ACTIVITY 3.9.6: Multiple if statements.

Consider the example above. What is the program's output for each value of userAge?

1) 16

Check Show answer

2) 50

Check Show answer

3) 13

Check Show answer

How was this section?

Provide feedback

3.10 Loops (general)

Loop concept

People who have children may be familiar with looping around the block until a baby falls asleep.

PARTICIPATION ACTIVITY 3.10.1: Loop concept: Driving a baby around the block.

Animation captions:

1. Parents may be familiar with this scenario: Driving home, baby is awake. Parents circle the block, hoping the baby will fall asleep.
2. After first loop, baby is still awake, so parents loop again.
3. After second loop, baby is asleep, so parents head home for a peaceful evening.

PARTICIPATION ACTIVITY 3.10.2: Loop concept.

Consider the example above.

1) When the parents first checked, was the baby awake?
   - Yes
   - No

2) After the first loop, was the baby awake?
   - Yes
   - No

3) After the second loop, was the baby awake?
   - Yes
   - No

4) How many loops around the block did the parents make?
   - 2
   -

3

5) Where was the decision point for whether to loop: At the top of the street or bottom?

- Top
- Bottom

Loop basics

A ***loop*** is a program construct that repeatedly executes the loop's statements (known as the ***loop body***) while the loop's expression is true; when false, execution proceeds past the loop. Each time through a loop's statements is called an ***iteration***.

PARTICIPATION ACTIVITY 3.10.3: A simple loop: Summing the input values.

Animation content:

```
A zyFlowchart program follows:
integer sum
integer val

sum = 0
val = Get next input

while val > -1
   sum = sum + val
   val = Get next input

Put sum to output

Input is as follows:
2 4 1 -1

Output (screen) is as follows:
7
```


Animation captions:

1. A loop is like a branch, but jumping back to the expression when done. Thus, the loop's statements may execute multiple times, before execution proceeds past the loop.

2. This program gets an input value. If the value > -1, the program adds the value to a sum, gets another input, and repeats. val is 2, so the loop's statements execute, making sum 2.
3. The loop's statements ended by getting the next input, which is 4. The loop's expression 4 > -1 is true, so the loop's statements execute again, making sum 2 + 4 or 6.
4. The loop's statements got the next input of 1. The loop's expression 1 > -1 is true, so the loop's statements execute a third time, making sum 6 + 1 or 7.
5. The next input is -1. This time, -1 > -1 is false, so the loop is not entered. Instead, execution proceeds past the loop, where a statement puts sum, which is 7, to the output.

Loop example: Computing an average

A loop can be used to compute the average of a list of numbers.

PARTICIPATION ACTIVITY 3.10.4: Loop example: Computing an average.

Animation content:

```
A zyFlowchart program follows:
integer sum
integer num
integer val

sum = 0
num = 0
val = Get next input

while val > -1
   sum = sum + val
   num = num + 1
   val = Get next input

avg = sum / num
Put sum to output

Input is as follows:
2 4 9 -1

Output (screen) is as follows:
5
```

Animation captions:

1. The program computes an average of a list of numbers (a negative ends the list). The first input is 2, so the loop is entered. Sum becomes 2, and num is incremented to 1.
2. The next input is 4. The loop is entered, so sum becomes 2 + 4 or 6, and num is incremented to 2.
3. The next input is 9, so the loop is entered. Sum becomes 6 + 9 or 15, and num is incremented to 3.
4. The next input is -1, so the loop is not entered. 15 / 3 or 5 is output.

PARTICIPATION ACTIVITY 3.10.5: Loop example: Average.

Consider the computing an average example above.

1) In the example above, the first value gotten from input was 2. That caused the loop body to be _____.
   - executed
   - not executed

2) At the end of the loop body, the _____.
   - next input is gotten
   - loop is exited
   - average is computed

3) With what value was sum initialized?
   - -1
   - 0

4) Each time through the loop, the sum variable is increased by _____.
   - 0
   - 1
   - the current input value

5) What was variable num's value after the loop was done iterating?
   - 1
   - 2

- 3

6) Before the loop, the first input value is gotten. If that input was negative (unlike the data in the example above), the loop's body would _____.

- be executed
- not be executed

Example: Counting specific values in a list

Programs execute one statement at a time. Thus, using a loop to examine a list of values one value at a time and updating variables along the way, as in the above examples, is a common programming task.

Below is a task to help a person get accustomed to examining a list of values one value at a time. The task asks a person to count the number of negative values, incrementing a variable to keep count.

PARTICIPATION ACTIVITY 3.10.6: Counting negative values in a list of values.

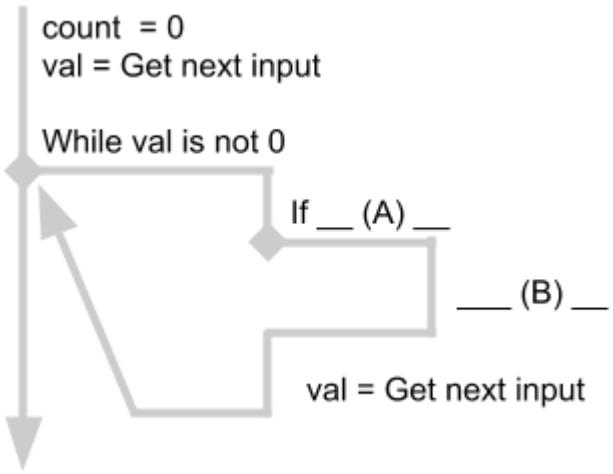
This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.10.7: Counting negative values.

Complete the program such that variable count ends having the number of negative values in an input list of values (the list ends with 0). So if the input is -1 -5 9 3 0, then count should end with 2.



1) What should expression (A) be?

- val > 0
- val < 0
- val is 0

2) What should statement (B) be?

- val = val + 1
- count = count + 1
- count = val

3) If the input value is 0, does the loop body execute?

- Yes
- No

Example: Finding the max value

Examining items one at a time and updating a variable can achieve some interesting computations. The task below is to find the maximum value in a list of positive values. A variable stores the max value seen so far. Each input value is compared with that max, and if greater, that value replaces that max. The max value is initialized with -1 so that such comparison works even for the first input value.

PARTICIPATION ACTIVITY 3.10.8: Find the maximum value in the list of values.

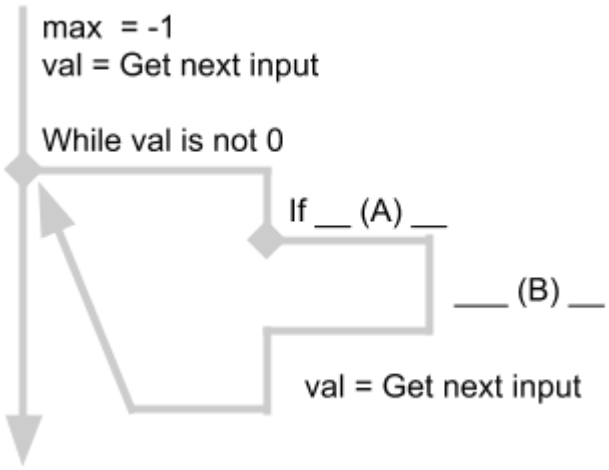
This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.10.9: Determining the max value.

Complete the program such that the program ends with variable max having the maximum value in an input list of positive values (the list ends with 0). So if the input is 22 5 99 3 0, then max should end as 99.



1) What should expression (A) be?

- max > 0
- max > val
- val > max

2) What should statement (B) be?

- max = val
- val = max
- max = max + 1

3) Does the final value of max depend on the order of inputs? In particular, would max be different for inputs 22 5 99 3 0 versus inputs 3 99 5 22 0?

- Yes
- No

4) For inputs 5 10 7 20 8 0, with what values will max be assigned.

- -1, 20
- -1, 5, 10, 20
- -1, 5, 10, 7, 20

How was this section?

Provide feedback

3.11 Loop basics

Loop basics

A ***loop*** is a program construct that repeatedly executes a list of sub-statements (known as the ***loop body***) while the loop's decision expression evaluates to true. A loop starts with a decision statement with the loops expression. If the expression is true, the loop body's statements are executed. At the loop body's end, execution proceeds to the loop's decision statement. Each execution of the loop body is called an ***iteration***. Once entering the loop body, execution continues to the body's end, even if the expression would become false midway through.

PARTICIPATION ACTIVITY 3.11.1: Loop basics.

Animation content:

```
A zyFlowchart program follows:
integer userNum
integer curPower

curPower = 0
userNum = Get next input

while userNum == 1
   Put curPower to output
   put "\n" to output
   curPower = curPower * 2
   userNum = Get next input

Put "Done." to output
```

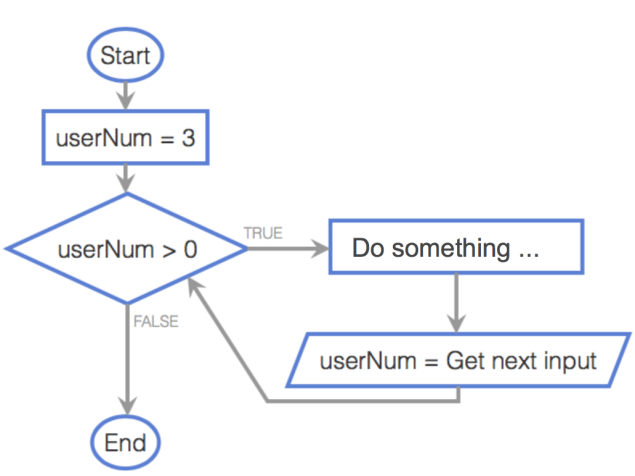
```
Variables in memory is as follows:
0 userNum: integer
8 curPower: integer

Input is as follows:
1 1 0

Output (screen) is as follows:
2
4
Done.
```


Animation captions:

1. A loop begins with a decision statement. The decision's expression is evaluated. If true, the loop's body is entered. Here, userNum is 1, so userNum == 1 is true.
2. Thus, the loop body is executed, which outputs curPower's current value of 2, doubles curPower, and gets the next input.
3. Execution jumps back to the loop's decision statement. userNum is 1 (the second input), so userNum == 1 is true, and the loop body executes (again), outputting 4.
4. userNum is now 0, so userNum == 1 is false. Thus, execution jumps to after the loop, which outputs "Done".

PARTICIPATION ACTIVITY 3.11.2: Loops: Number of iterations.

For the flowchart below, indicate how many times the loop body will execute for the indicated input values.



Variables

0	userNum	integer

1) Input: 5 -1

Check Show answer

2) Input: 2 1 0

Check Show answer

3) Input: -1 5 4

Check Show answer

Looping until done

Programmers often use loops to execute a computation until a done condition is reached. That done condition is reached when the loop expression is false. The following Celsius to Fahrenheit example loops until the user input is not 1. The example shows the common pattern of getting user input at the end of each loop iteration to determine whether to continue looping.

PARTICIPATION ACTIVITY 3.11.3: Loop example: Celsius to Fahrenheit.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.11.4: Looping until done: Celsius to Fahrenheit.

Consider the example above.

1) The loop will always iterate at least once.

- True
- False

2) For the user input provided, how many times did the loop iterate?

- 9
- 10

3) Each iteration adds _____ to celsiusValue.

- 1
- 5

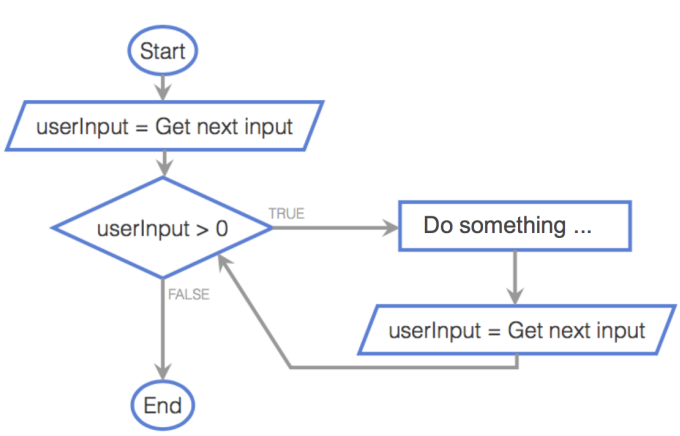
4) If the user's first input is 0, how many times will the loop iterate?

- 1
- 2

Getting input before a loop

The above examples got user input into a variable only at the end of the loop body. The examples assigned that variable an initial value that always caused the loop body to execute the first time. Another common pattern gets that initial value from user input as well, thus getting input in two places: before the loop, and at the loop body's end.

Figure 3.11.1: Common pattern: Getting input before and at end of loop.



Variables

0	userInput	integer

PARTICIPATION ACTIVITY 3.11.5: Looping until done with user input before the loop.

For the flowchart in the figure above, indicate how many times the loop body will execute for the indicated input values.

1) Input: 5 -1

Check Show answer

2) Input: 2 1 0

Check Show answer

3) Input: -1 5 4

Check Show answer

Loop expressions

Various kinds of expressions are found in loop expressions. For example, sometimes a loop is executed as long as a value is greater-than another value, or less-than another value. Sometimes a loop is executed as long as a value is NOT equal to another value.

Below is an example with a relational operator in the loop expression. The program prints a horizontal bar chart for the user input, with each "-" representing 10 units.

PARTICIPATION ACTIVITY 3.11.6: Loop using a relational operator in the loop expression.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.11.7: Loop expressions.

Use a *single operator* in each loop expression, and the most straightforward translation of the stated goal into an expression.

1) Iterate while xVal is less-than 100.

Check Show answer

2) Iterate while xVal is greater than or equal to 0.

Check Show answer

3) Iterate while cVal equals 5.

Check Show answer

4) Iterate while cVal is not equal to 10.

Check Show answer

5) Iterate *until* cVal equals 15 (tricky; think carefully).

Check Show answer

Example: Ancestors

Below is another loop example. The program asks the user to enter a year, and then outputs the approximate number of a person's ancestors who were alive for each generation leading back to that year, with the loop computing powers of 2 along the way.

PARTICIPATION ACTIVITY 3.11.8: Loop example: Ancestors printing program.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Each iteration prints a line with the year and the ancestors in that year. (Note: the numbers are large due to not considering breeding among distant relatives, but nevertheless a person has many ancestors).

PARTICIPATION ACTIVITY 3.11.9: Ancestors example.

Consider the example above.

1) The loop expression involves a relational operator.
   - True
   - False

2) The loop body updates the current year curYear.
   - True
   - False

3) The user is asked to enter a value at the end of each loop iteration.
   - True
   - False

4) Each loop iteration outputs the current number of ancestors (numAncestors), and then doubles numAncestors in preparation for the next iteration.
   - True
   - False

Common errors

A common error is to use the opposite loop expression than desired, like using x == 0 rather than x != 0. Programmers should remember that the expression describes when the loop *should* iterate, not when the loop should exit.

An ***infinite loop*** is a loop that never stops iterating. A common error is to accidentally create an infinite loop, often by forgetting to update a variable in the body, or by creating a loop expression whose evaluation to false isn't always reachable.

PARTICIPATION ACTIVITY 3.11.10: Infinite loop.

Animation content:

```
A zyFlowchart program follows:
integer numKids
integer userNum

numKids = 2
userNum = Get next input

while userNum == 1
   Put numKids to output
   numKids = numKids * 2
   // Oops, forgot to get userNum from input again.
```

Animation captions:

1. This loop gets userNum from input, iterating if userNum is 1. If the first input is 1, the loop body executes a first time.
2. The loop body outputs numKids and updates numKids. But, the programmer forgot to get userNum from the input at the end of the loop body.
3. Thus, userNum is still 1, and the loop body is executed again, and again, and again, with no way out. The loop is an "infinite loop".

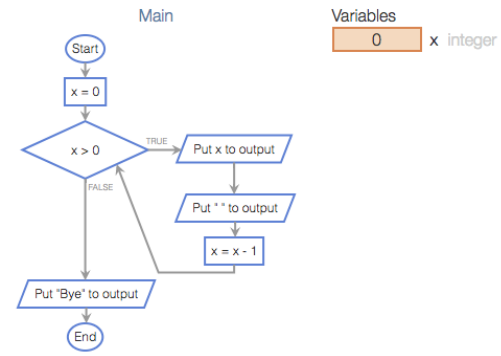
For the above example, a programmer must get in the habit of remembering to update needed variables at the loop body's end.

A program with an infinite loop may print excessively, or just seem to stall. In the zyFlowchart interpreter, the user can halt execution by selecting Pause or Exit Execution.

PARTICIPATION ACTIVITY 3.11.11: Loop iterations.

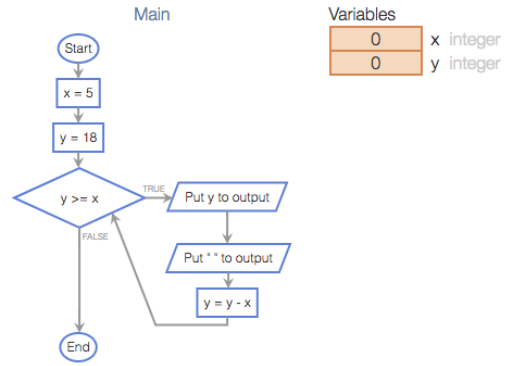
What will the following code output? For an infinite loop, type "IL" (without the quotes).

1)



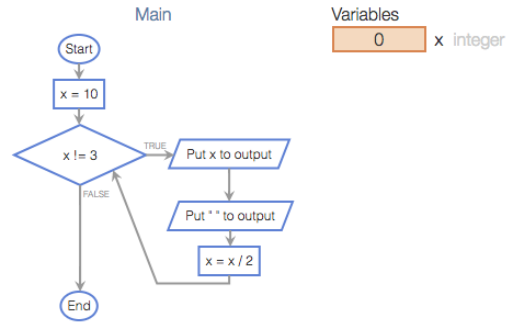
Check Show answer

2)



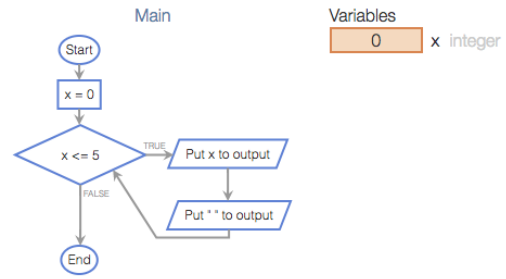
Check Show answer

3)



Check Show answer

4)



Check Show answer

How was this section?

Provide feedback

3.12 More loop examples

Example: GCD

The following is an example of using a loop to compute a mathematical quantity. The program computes the greatest common divisor (GCD) among two user-entered integers numA and numB, using Euclid's algorithm: If numA > numB, set numA to numA - numB, else set numB to numB - numA. These steps are repeated until numA equals numB, at which point numA and numB each equal the GCD.

PARTICIPATION ACTIVITY 3.12.1: Loop example: GCD (greatest common divisor) program.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.12.2: GCD program.

Refer to the GCD program above. Assume user input of numA = 15 and numB = 10.

1) For the GCD program, what is the value of numA *before* the first loop iteration?

Check Show answer

2) What is the value of numB *after* the first iteration of the loop?

Check Show answer

3) What is numB after the second iteration of the loop?

Check Show answer

4) How many loop iterations will the algorithm execute?

Check Show answer

Example: Getting input until a sentinel is seen

Loops are commonly used to process an input list of values. A ***sentinel value*** is a special value indicating the end of a list, such as a list of positive integers ending with 0, as in 10 1 6 3 0. The example below computes the average of an input list of positive integers, ending with 0. The 0 is not included in the average.

PARTICIPATION ACTIVITY 3.12.3: Computing average of a list with a sentinel.

This activity failed to load. Please

try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.12.4: Average example with a sentinel.

Consider the example above.

1) How many actual (non-sentinel) values are given in the input sequence?
   - 1
   - 4
   - 5

2) For the input sequence, what is the final value of numValues?
   - 0
   - 4
   - 5

3) Suppose the first input was 0. Would (1.0 * valuesSum / numValues) be 0?
   - Yes
   - No

4) What would happen if this list was input: 10 1 6 3 -1
   - Output would be 5
   - Output would be 4
   - Error

How was this section?

Provide feedback

3.13 Looping N times

Looping N times

A loop commonly must iterate a specific number of times, such as 10 times. A loop iterating a specific number of times commonly consists of three parts: a loop variable initialization before the loop, a decision statement for the loop expression, and a loop variable update at the end of the loop body.

PARTICIPATION ACTIVITY 3.13.1: Looping N times.

Animation content:

```
A zyFlowchart program follows:
integer i

// Loop variable initialization
i = 0

// Loop's expression
while i < 3

   // Loop's statements

   // Loop variable update
   i = i + 1

Variables in memory as follows:
3 i: integer
```

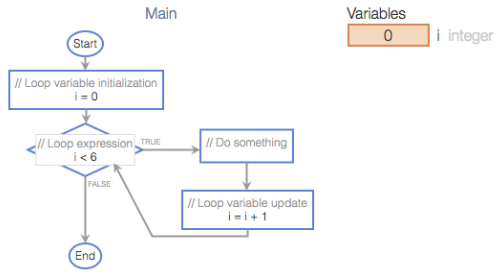

Animation captions:

1. A loop that iterates a specific number of times consists of three parts: a loop variable initialization, a loop expression, and a loop variable update.
2. A loop that iterates 3 times, with i = 0, 1, and 2, initializes i with 0, uses the loop expression i < 3, and ends the loop body with i = i + 1.

3. The loop iterates when i is 0, 1, and 2. After updating i to 3, the loop exits.

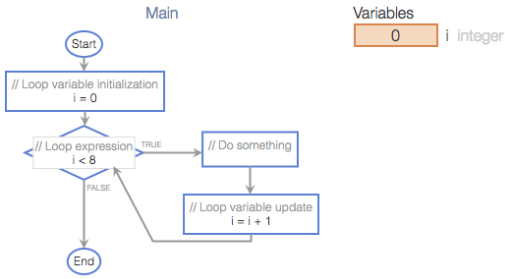
PARTICIPATION ACTIVITY 3.13.2: Looping N times.

1) What are the values of i for each iteration of:



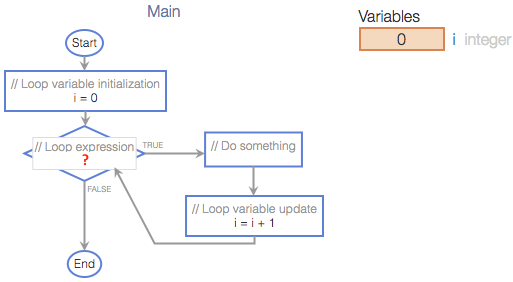
- 1, 2, 3, 4, 5
- 0, 1, 2, 3, 4, 5
- 0, 1, 2, 3, 4, 5, 6

2) How many times will this loop iterate?



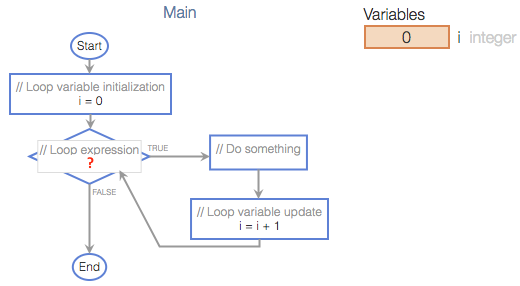
- 7 times
- 8 times
- 9 times

3) Goal: Loop 10 times



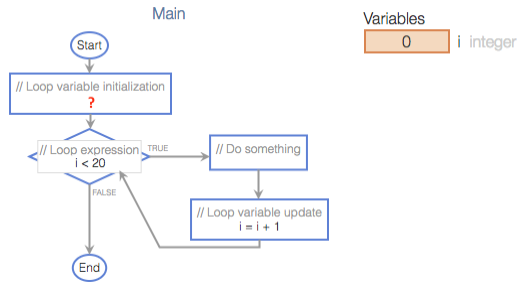
- i < 9
- i < 10
- i < 11

4) Goal: Loop 99 times



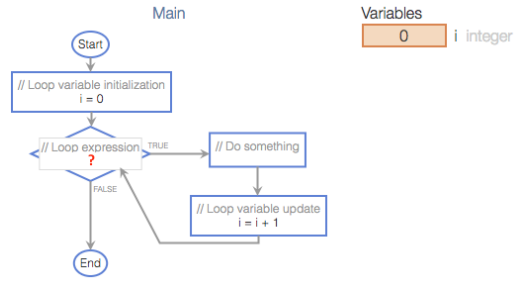
- i < 99
- i <= 99
- i == 99

5) Goal: Loop 20 times



- i = 0
- i = 1

6) Goal: Loop numYears times (numYears is an integer variable).



- numYears
- i <= numYears
- i < numYears

Example: Savings with interest

The following program outputs the amount of a savings account each year for 10 years, given an input initial amount and interest rate. The loop iterates 10 times, such that each iteration represents one year, outputting that year's savings amount.

PARTICIPATION

ACTIVITY 3.13.3: Loop N times example: Savings interest program.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.13.4: Savings interest program.

Consider the example above.

1) How many times does the loop iterate?
   - 5
   - 10

2) During each iteration, the loop body's statements output the current savings amount, and then _____.
   - increment i
   - update currSavings

3) Can the input values change the number of loop iterations?
   - Yes
   - No

Example: Computing the average of a list of input values

The example below computes the average of an input list of integer values. The first input indicates the number of values in the subsequent list. That number controls how many times the subsequent loop iterates.

PARTICIPATION ACTIVITY 3.13.5: Computing an average, with first value indicating list size.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists, **send feedback to zyBooks support**

PARTICIPATION ACTIVITY 3.13.6: Computing the average.

Consider the example above, with input 4 10 1 6 3.

1) Before the loop is entered, what is valuesSum?

Check Show answer

2) What is valuesSum after the first iteration?

Check Show answer

3) What is valuesSum after the second iteration?

Check Show answer

4) valuesSum is 20 after the fourth iteration. What is numValues?

Check Show answer

5) For the following input, how many

times will the loop iterate? 7 -1 -3 -5 -14 -15 -20 -40

Check Show answer

How was this section?

Provide feedback

3.14 Loop examples iterating N times

Example: Finding the max

Analyzing data is a common programming task. A common data analysis task is to find the maximum value in a list of values. A loop can achieve that task by updating a max-seen-so-far variable on each iteration.

PARTICIPATION ACTIVITY 3.14.1: Computing a maximum, with first value indicating list size.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.14.2: Finding the max.

Consider the example above.

1) Before entering the loop, what is the maximum value seen so far from

the list of integers?

- 0
- -1
- No such value

2) The loop's first iteration gets the list's first integer into variable currValue. Is that the maximum value seen so far?

- Yes
- No

3) For each iteration after the first iteration, the comparison _____ is checked.

- maxSoFar > currValue
- currValue > maxSoFar
- currValue == maxSoFar

Beyond iterating N times

By adjusting the three parts of a loop iterating N times, a loop can output various sequences. The following outputs multiples of 5 from 10 to 50.

PARTICIPATION ACTIVITY 3.14.3: Outputting multiples of 5 from 10 to 50.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

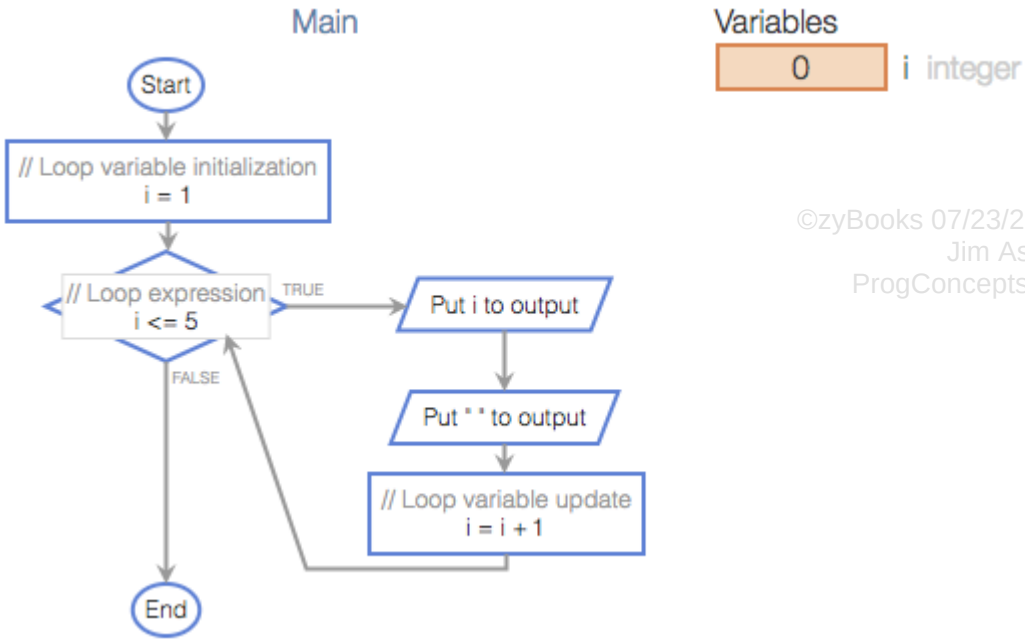
If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.14.4: Beyond iterating N times.

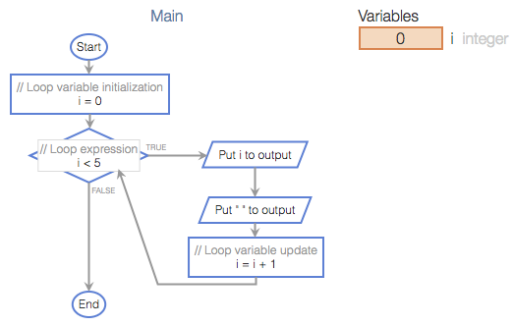
Type the output of the loop. Whitespace matters, including after the last item.

Example:



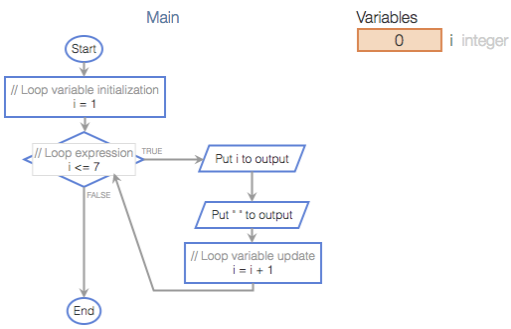
Outputs:
1 2 3 4 5

1)



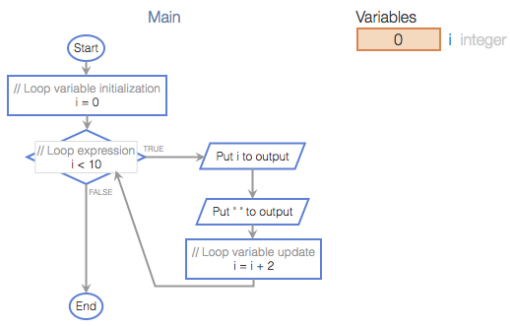
Check Show answer

2)



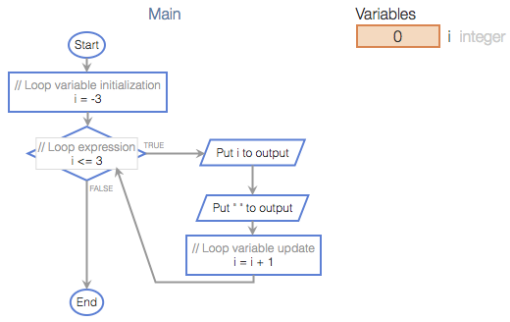
Check Show answer

3)



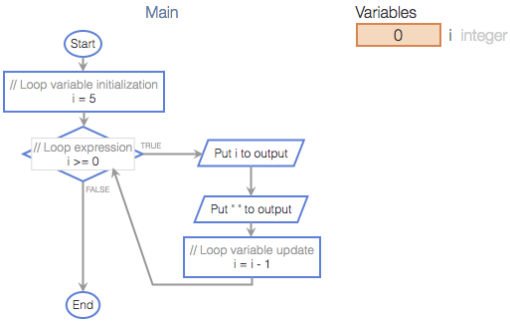
Check Show answer

4)



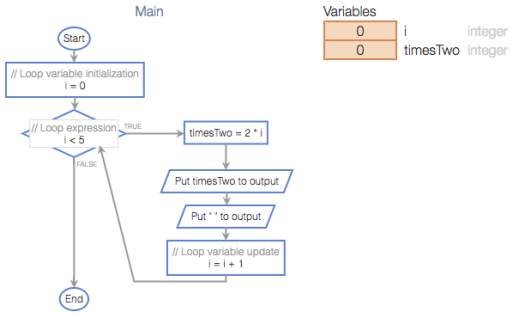
Check Show answer

5)



Check Show answer

6)



Check Show answer

Example: Outputting a score sheet

Programs are sometimes used to auto-generate data tables. The following program generates a table of Celsius and Fahrenheit temperature values, in increments of 5 C. The loop counts from -10 to 40 in increments of 5, and names the loop variable currC rather than i to be more descriptive.

PARTICIPATION ACTIVITY 3.14.5: Auto-generate a data table: Celsius to Fahrenheit.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 3.14.6: Loop generating a table of temperature values.

Consider the example above.

1) What is the loop variable's name?

Check Show answer

2) What are the values of currC for the first four iterations?
Type as: 1 9 2 6

Check Show answer

3) What is the loop expression? (The expression checked for whether to enter the loop body).

Check Show answer

CHALLENGE ACTIVITY 3.14.1: Loop iterating N times and beyond N times.

How was this section?

Provide feedback

3.15 While and for loops

while and for loops

Because looping until done and looping N times are so common, most textual programming languages have specific constructs for these types of loops. A ***while loop*** is a loop that repeatedly executes the loop body *while* the loop's expression evaluates to true. A ***for loop*** is a loop consisting of a loop variable initialization, a loop expression, and a loop variable update that typically describes iterating *for* a specific number of times.

Generally, a programmer uses a for loop when the number of iterations is known (like loop 5 times, or loop numItems times), and a while loop otherwise.

Table 3.15.1: Choosing between while and for loops: General guidelines (not strict rules though).

for	Number of iterations is computable before the loop, like iterating N times.
while	Number of iterations is not (easily) computable before the loop, like iterating until the input is 'q'.

PARTICIPATION ACTIVITY 3.15.1: While loops and for loops.

Choose the most appropriate loop type.

1) Iterate as long as user-entered integer c is not 0.
   - while
   - for

2) Iterate until the values of x and y are equal, where x and y are changed in the loop body.
   - while
   - for

3) Iterate 100 times.
   - while
   - for

How was this section?

Provide feedback

3.16 Nested loops

A ***nested loop*** is a loop that appears in the body of another loop. The nested loops are commonly referred to as the ***inner loop*** and ***outer loop***.

Nested loops have various uses. Below is a nested loop example that graphically depicts an integer's magnitude by using asterisks, creating a bar chart. The inner loop is a for loop that handles the printing of the asterisks. The outer loop is a while loop that handles executing until a negative number is entered.

PARTICIPATION ACTIVITY 3.16.1: Nested loop example: Bar chart.

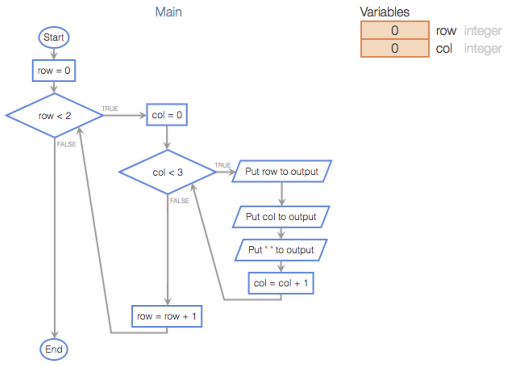
This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

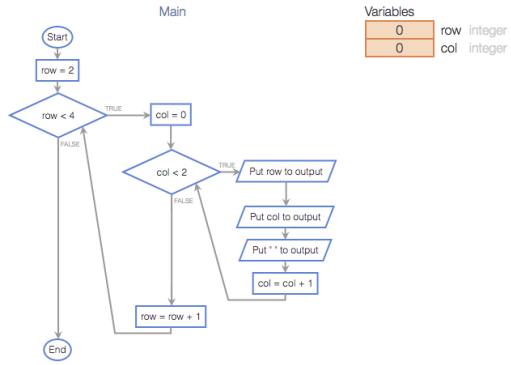
PARTICIPATION ACTIVITY 3.16.2: Nested loops: Inner loop execution.

1) Given the following code, how many times will the inner loop body execute?



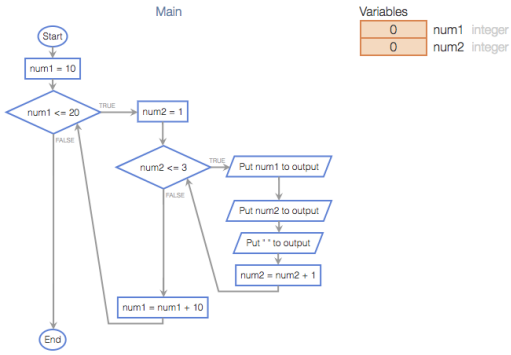
Check Show answer

2) What is output by the following code?



Check Show answer

3) What is output by the following code?



Check Show answer

How was this section?

Provide feedback

3.17 Do-while loop

A ***do-while loop*** is a loop that first executes the loop body's statements, then checks the loop condition. Compared to a while loop, a do-while loop is useful when the loop should iterate at least once.

PARTICIPATION ACTIVITY 3.17.1: Do-while loop.

Animation content:

```
A zyFlowchart program follows:
integer curCount
integer userNum

curCount = 0
userNum = Get next input

do
   Put curCount to output
   userNum = userNum - 1
   curCount = curCount + 1
while userNum > 4

Input is as follows:
6

Output (screen) is as follows:
0 1
```

Animation captions:

1. The loop body's statements first, then checks the loop condition.
2. Because userNum > 4, a second iteration will occur.
3. Because 4 > 4 evaluates to false, execution proceeds past the loop.

PARTICIPATION ACTIVITY 3.17.2: Do-while loop.

Refer to the loop in the animation above:

1) What is the value of curCount after the loop?

- 0
-

1

2

2) What user input would prevent curCount from being incremented?

4

0

No such value.

How was this section?

Provide feedback

3.18 Pseudocode: Loops

while loops

In pseudocode, a loop can be written as a ***while loop***: A while loop repeatedly executes the loop body while the loop's expression evaluates to true.

PARTICIPATION ACTIVITY 3.18.1: Pseudocode for while loop.

Animation captions:

1. In pseudocode, a while loop uses the word "while" followed by the expression.
2. The loop body's statements start on the next line and are indented.
3. If the expression is true, the loop body executes. After executing the loop body, execution jumps back to the while loop's condition.
4. When the loop expression is false, execution proceeds to the statement after the loop.

PARTICIPATION ACTIVITY 3.18.2: While loop pseudocode.

Use a *single operator* in each loop expression, and the most straightforward translation of the stated goal into an expression.

```
while _____
   // Loop statements
Put "Done." to output
```

1) Iterate while x is less than 100.

Check Show answer

2) Iterate while x is greater than or equal to 0.

Check Show answer

3) Iterate while userNum is not -1.

Check Show answer

for loop

In pseudocode, a loop that iterates a specific number of times is typically implemented using a for loop. A ***for loop*** is a loop with three parts at the top: a loop variable initialization, a loop expression, and a loop variable update. A for loop describes iterating a specific number of times more naturally than a while loop.

PARTICIPATION ACTIVITY 3.18.3: For loop looping N times.

Animation captions:

1. A for loop uses the word "for" followed by the loop variable initialization, the loop condition, and the loop variable update. Note that semicolons separate the three parts. No semicolon is needed at the end.
2. The loop body's statements, not including the loop variable update, start on the next line and are indented.
3. The loop variable initialization executes once at the beginning of the loop execution. If the loop condition is true, the loop's statements execute.
4. At the end of the loop execution, the loop variable update statement is executed, and then the loop condition is checked again.
5. When the loop expression is false, execution proceeds to the statement after the loop.

PARTICIPATION ACTIVITY 3.18.4: For loops.

1) What are the values of i for each iteration of:

```
for i = 0; i < 6; i = i + 1
   // Loop statements
```

- 1, 2, 3, 4, 5
- 0, 1, 2, 3, 4, 5
- 0, 1, 2, 3, 4, 5, 6

2) How many times will this loop iterate?

```
for i = 0; i < 8; i = i + 1
   // Loop statements
```

- 7 times
- 8 times
- 9 times

3) Goal: Loop 10 times

```
for i = 0; _____; i = i + 1
   // Loop statements
```

- i < 9
- i < 10
- i < 11

4) Goal: Loop 99 times

```
for i = 0; _____; i = i + 1
   // Loop statements
```

- i < 99
- i <= 99
- i == 99

5) Goal: Loop 20 times

```
for ____; i < 20; i = i + 1
   // Loop statements
```

- i = 0
- i = 1

6) Goal: Loop numYears times (numYears is an integer variable)

```
for i = 0; _____; i = i + 1
   // Loop statements
```

- numYears
- i <= numYears
- i < numYears

do-while loop

A ***do-while*** loop is a loop that first executes the loop body's statements, then checks the loop condition.

Figure 3.18.1: Do-while loop pseudocode.

```
integer curCount
integer userNum

curCount = 0
userNum = Get next input

do
   Put curCount to output
   userNum = userNum - 1
   curCount = curCount + 1
while userNum > 4
```

PARTICIPATION ACTIVITY 3.18.5: Do-while loop pseudocode.

For the following pseudocode, indicate what is output for the given input values.

```
integer userNum

userNum = Get next input

do
   Put userNum to output
   Put " " to output
   userNum = userNum + 1
while userNum < 10
```

1) Input: 8

- 8
- 8 9
- 8 9 10

2) Input: 5

- 5 6 7 8 9
- 5 6 7 8 9 10

3) Input: 10

- Nothing
- 10

How was this section?

Provide feedback

3.19 Control structures: Branches summary

This chapter's key points included:

- In a flowchart, a decision creates two branches, one for when the decision's expression is true (the if branch), another when false (the else branch).
- If-else branches have statements in each branch. An if branch has no statements in the else branch.
- If-elseif branches have cascaded decisions along the false branches. Only one true branch can execute.
- A branch can itself have a decision, known as nested branches.
- Multiple if branches can be created, which are independent, so more than one true branch can execute.
- Valid equality and relational operators are ==, !=, <, <=, >, >=.
- If-elseif branches are commonly used to detect ranges, with the lower end of the range implicit.
- Logical operators are: and, or, not.
- In an expression, operators are evaluated in a specific order based on precedence rules (just like in math).
- Because floating-point numbers aren't represented exactly, they shouldn't be compared for equality (using ==). Instead, they can be compared for "close enough".
- Branches in pseudocode use words like if, else, and elseif, and a branch's statements appear indented starting on a next line.

PARTICIPATION ACTIVITY 3.19.1: Choosing the appropriate if statement type.

Which control structure should be used for each situation?

1) A program should output "Too heavy" if weight is over 200 pounds.

Otherwise, the program outputs nothing.

- If
- If-else
- If-elseif

2) A program should output "Too heavy" if weight is over 200, and output "OK" otherwise.

- If
- If-else
- If-elseif

3) A program should output "Too heavy" if weight is over 200, "Warning" if weight is over 180, and "OK" otherwise.

- If
- If-else
- If-elseif

4) A program should output "Warning" for any weight over 180. Additionally, the program should output "Too heavy" if weight is over 200.

- If-else
- If-elseif
- Multiple if's

PARTICIPATION ACTIVITY 3.19.2: Logical and relational operators.

1) Which is the standard operator for greater-than-or-equal?

- >=
- =>

2) Which logical operator should be used so that this expression is true

only if x is between 10 and 20:

```
(x > 10) ___ (x < 20)
```

- and
- or

PARTICIPATION ACTIVITY 3.19.3: Determining the output of branches.

What is put to output by each program?

1)

```
x = 30
if x < 20
   Put "Small" to output
else
   Put "Large" to output
```

- Small
- Large
- (no output)

2)

```
x = 30
if x < 20
   Put "Small" to output
elseif x < 30
   Put "Large" to output
else
   Put "Huge" to output
```

- Small
- Large
- Huge

3)

```
x = 15
if x < 20
   Put "A" to output

if x < 30
   Put "B" to output
```

- A
- B
- AB

How was this section?

Provide feedback

3.20 Control structures: Loops summary

This chapter's key points included:

- A loop is a program construct that repeatedly executes the loop's statements (known as the loop body) while the loop's expression is true; when false, execution proceeds past the loop. Each time through a loop's statements is called an iteration.
- A common programming task is to use a loop to examine a list of values one value at a time and update variables along the way. Common tasks include computing an average, counting the number of negative items in a list, finding the maximum, etc..
- Programmers often use loops to execute a computation until a done condition is reached. That done condition is reached when the loop expression is false.
- An infinite loop is a loop that never stops iterating.
- A loop iterating a specific number of times commonly consists of three parts: a loop variable initialization before the loop, a decision statement for the loop expression, and a loop variable update at the end of the loop body.
- A while loop is a loop that repeatedly executes the loop body while the loop's expression evaluates to true.
- A for loop is a loop that typically describes iterating for a specific number of times. A for loop consists of a loop variable initialization, a loop expression, and a loop variable update statement.
- A nested loop is a loop that appears in the body of another loop. The nested loops are commonly referred to as the inner loop and outer loop.
- A do-while loop is a loop that first executes the loop body's statements, then checks the loop condition. Compared to a while loop, a do-while loop is useful when the loop should iterate at least once.

PARTICIPATION ACTIVITY 3.20.1: Loops summary.

1) A program should output even numbers from 2 to 50. Which control structure should be used?

   - ○ if statement
   - ○ while loop
   - ○ for loop

2) A program should continue getting an input value, and converting that value from pounds to kilograms, until the input is zero. Which control structure should be used?

- if statement
- while loop
- for loop

3) A program first assigns userNum with a value read from user input. The program outputs the value of userNum and adds 1 to userNum. Then, if userNum is less than 100, the program repeats, outputting the value of userNum and adding 1 to userNum. Which control structure should be used?

- while loop
- for loop
- do-while loop

4) What is put to output by the following pseudocode?

```
x = 10
while x > 4
   Put x to output
   Put " " to output
   x = x - 2
```

- 10 9 8 7 6 5
- 10 8 6 4
- 10 8 6

5) What is the loop condition expression in the following pseudocode?

```
i = 5
while i <= 25
   Put i to output
   i = i + 1
```

- i
- i = 5
- i <= 25

6) What is the loop variable initialization in the following

pseudocode?

```
x = 0
y = 10
while y > 0
   y = y - 1
   x = x + y
```

- x = 0
- y = 10
- y > 0

7) What is the loop variable update statement in the following pseudocode?

```
x = 1
y = 2
while x != 10
   Put x to output
   x = x + 1
```

- x = 1
- x != 10
- x = x + 1

8) What is put to output by the following pseudocode?

```
for i = 1; i < 4; i = i + 1
   Put i to output
   Put " " to output
```

- 0 1 2 3 4
- 1 2 3 4
- 1 2 3

9) What is put to output by the following pseudocode?

```
x = 10
do
   x = x - 4
   Put x to output
   Put " " to output
while x > 0
```

- 6 2 -2
- 6 2
- 10 6 2

How was this section?

Provide feedback

©zyBooks 07/23/20 10:53 175839
Jim Ashe
ProgConceptsWGUC173

4.1 User-defined function basics

Basics of functions

A ***function*** is a named list of statements.

- A ***function definition*** consists of the new function's name and a block of statements. The function's name can be any valid identifier.
- A ***function call*** is an invocation of a function's name, causing the function's statements to execute.

A program's execution begins with the Main function. Below, the function call PrintPizzaArea() causes execution to jump to the function's statements. Execution jumps back to the original location after executing the function's last statement.

PARTICIPATION ACTIVITY 4.1.1: Function example: Printing a pizza area.

Animation content:

```
A zyFlowchart program follows:
Function nothing PrintPizzaArea()
   float piVal
   float pizzaDiameter
   float pizzaRadius
   float pizzaArea

   piVal = 3.14159265
   pizzaDiameter = 12.0
   pizzaRadius = pizzaDiameter / 2.0
   pizzaArea = piVal * pizzaRadius * pizzaRadius
   Put pizzaDiameter to output
   Put " in. pizza is " to output
   Put pizzaArea to output
   Put " sq. in.\n" to output

Function nothing Main()
   PrintPizzaArea()

Local variables in memory are as follows:
```

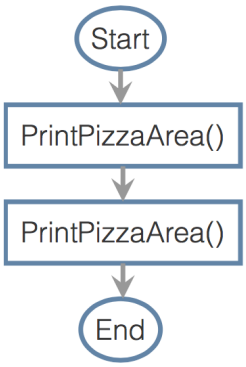
```
3.14159265 piVal: float
12.0 pizzaDiameter: float
6.0 pizzaRadius: float
113.097 pizzaArea: float

Output (screen) is as follows:
12.0 in. pizza is 113.097 sq. in.
```


Animation captions:

1. The function call jumps execution to the function's statements.
2. When the function's End statement is reached, execution jumps back to original call.

PARTICIPATION ACTIVITY 4.1.2: Function basics.

Given the PrintPizzaArea function defined above and the following Main function:



1) How many function calls to PrintPizzaArea exist in Main?

Check Show answer

2) How many function definitions of PrintPizzaArea exist *within* Main?

Check Show answer

3) How many output statements would execute in total?

Check Show answer

Check Show answer

4) How many output statements exist in PrintPizzaArea?

Check Show answer

5) Is Main itself a function definition? Answer yes or no.

Check Show answer

Parameters

A programmer can influence a function's behavior via an input.

- A ***parameter*** is a function input specified in a function definition. Ex: A pizza area function might have diameter as an input.
- An ***argument*** is a value provided to a function's parameter during a function call. Ex: A pizza area function might be called as PrintPizzaArea(12.0) or as PrintPizzaArea(16.0).

A parameter is like a variable declaration. Upon a call, the parameter's memory location is allocated, and the parameter is assigned with the argument's value. Upon return, the parameter is deleted from memory.

An argument may be an expression, like 12.0, x, or x * 1.5.

PARTICIPATION ACTIVITY | 4.1.3: Function with parameters example: Printing a pizza area for different diameters.

Animation content:


```
A zyFlowchart program follows:
Function nothing PrintPizzaArea(float pizzaDiameter)
   float piVal
   float pizzaRadius
   float pizzaArea

   piVal = 3.14159265
   pizzaRadius = pizzaDiameter / 2.0
```

```
   pizzaArea = piVal * pizzaRadius * pizzaRadius
   Put pizzaDiameter to output
   Put " in. pizza is " to output
   Put pizzaArea to output
   Put " sq. in.\n" to output

Function nothing Main()
   PrintPizzaArea()

Parameter variables in memory are as follows:
(empty) pizzaDiamater: float

Local variables variables in memory are as follows:
(empty) piVal: float
(empty) pizzaRadius: float
(empty) pizzaArea: float

Output (screen) is as follows:
12.0 in. pizza is 113.097 sq. in.
16.0 in. pizza is 201.062 sq. in.
```


Animation captions:

1. pizzaDiameter is a function parameter, for which an argument will be passed when the function is called.
2. The function call jumps execution to the function's statements, passing 12.0 to the function's pizzaDiameter parameter.
3. The next function call passes 16.0 to the function's pizzaDiameter parameter, which results in a different pizza area.

PARTICIPATION ACTIVITY 4.1.4: Parameters.

1) Write a statement that calls a function named PrintAge, passing the value 21 as an argument.

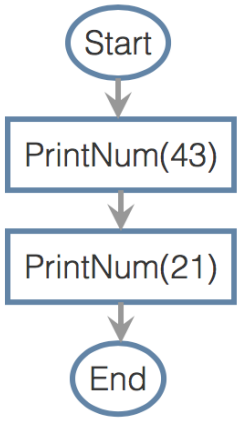
Check Show answer

2) Is the following a valid function call? Type yes or no.

```
MyFct(integer userNum + 5)
```

Check Show answer

3) Assume a function PrintNum simply prints the value of a parameter userNum without any space or new line. What will the following output?



Check Show answer

Multiple or no parameters

A function definition may have multiple parameters. Parameters are assigned with argument values by position: First parameter with first argument, second with second, etc. In Coral flowcharts, function parameters are listed from top to bottom.

A function may also be defined with no parameters. The call must include parentheses, with no argument, as in: `PrintSomething()`.

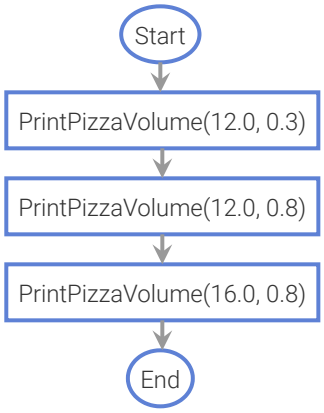
PARTICIPATION ACTIVITY 4.1.5: Function with multiple parameters. Full screen

Main | PrintPizzaVolume

Variables

None

Output



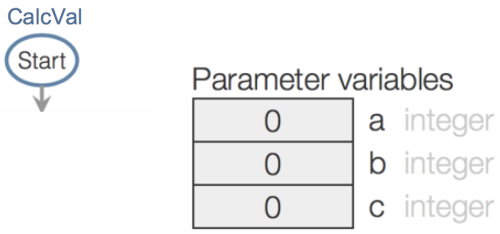
ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 4.1.6: Multiple parameters.

1) Which correctly passes two integer arguments for the function call: CalcVal(...)?

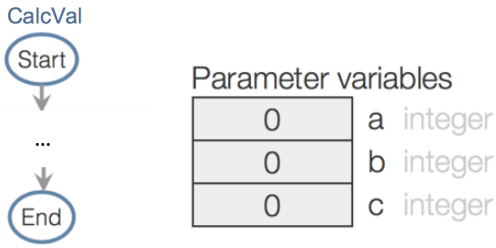
- (99, 44 + 5)
- (integer 99, 44)
- (integer 99, integer 44)

2) Given the CalcVal function definition below, b is assigned with what value during the function call: CalcVal(42, 55, 77)



- Unknown
- 42
- 55

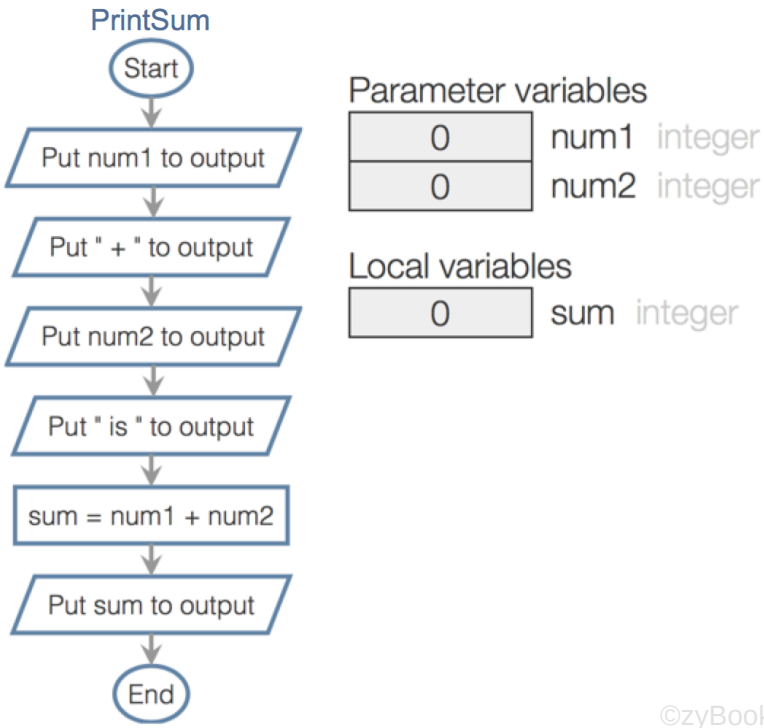
3) Given the CalcVal function definition below and integer variables i, j, and k, which are valid arguments in the call CalcVal(...)?



- (i, j)
- (k, i + j, 99)
- (i + j + k)

PARTICIPATION ACTIVITY 4.1.7: Calls with multiple parameters.

Given:



1) What will be printed for the following function call?

```
PrintSum(1, 2)
```

Check Show answer

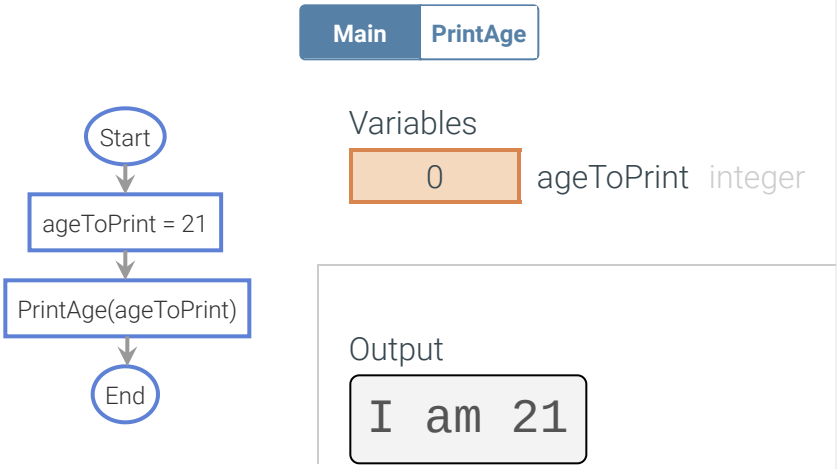
2) Write a statement that calls PrintSum() to print the sum of x and 400 (providing the arguments in that order).

Check Show answer

CHALLENGE ACTIVITY 4.1.1: Function basics.

Start

Type the program's output



1 2 3 4

Check Next

How was this section?

Provide feedback

4.2 Return

Returning a value from a function

A function may return one value by assigning a ***return variable*** with the ***return value***. Below, the ComputeSquare() function is defined to return the integer value held in the return variable numSquared. When the function reaches the End statement, the value held in numSquared is returned.

PARTICIPATION ACTIVITY 4.2.1: Function returns computed square.

Animation content:

```
A zyFlowchart program follows:
Function integer numSquared ComputeSquare(integer numToSquare)
   numSquared = numToSquare * numToSquare

Function nothing Main()
   integer squaredVal

   squaredVal = ComputeSquare(7)
   Put squaredVal to output

Variables in memory are as follows:
49 squaredVal: integer

Parameter variables in memory are as follows:
7 numToSquare: integer

Return variable in memory are as follows:
49 numSquared: integer

Output (screen) is as follows:
49
```

Animation captions:

1. The ComputeSquare is defined with a return variable numSquared of type integer.
2. ComputeSquare is called, passing 7 to the function's numToSquare parameter.
3. The function assigns the return variable numSquared with the square of the parameter numToSquare.
4. When the function execution completes, the value held numSquared is returned.ComputeSquare(7) evaluates to 49, and squaredVal is assigned with that value.

A function can only return one item, not two or more. A function that does not return a value will not declare a return variable.

PARTICIPATION ACTIVITY 4.2.2: Return value.

Refer to the animation above.

1) What is the return variable's name?
   - ComputeSquare
   - numSquared
   - numToSquare

2) What value is returned by the function call ComputeSquare(9)?
   - 3
   - 9
   - 81

3) Given an integer variable sumOfSquares, what value is held in sumOfSquares after the following statement?

   sumOfSquares = ComputeSquare(2) + ComputeSquare(6)
   - 4
   - 36
   - 40

4) A function can have two return variables.
   - True
   - False

Calling functions in expressions

A function call evaluates to the returned value. Thus, a function call often appears within an expression. Ex: `5 + ComputeSquare(4)` evaluates to 5 + 16, or 21.

A function that does not return a value cannot be used within an expression, instead being used in a statement like: `OutputData(x, y)`

PARTICIPATION ACTIVITY 4.2.3: Calls in an expression.

Given a float variable x, the built-in SquareRoot function, and a PrintVal function with a float parameter and no return value, which are valid statements?

1) `y = SquareRoot(49.0)`
   - True
   - False

2) `SquareRoot(49.0) = z`
   - True
   - False

3) `y = 1.0 + SquareRoot(144.0)`
   - True
   - False

4) `y = SquareRoot(SquareRoot(16.0))`
   - True
   - False

5) `y = SquareRoot()`
   - True
   - False

6) `SquareRoot(9.0)`
   - True
   - False

7) `y = PrintVal(9.0)`
   - True
   - False

8) `PrintVal(9.0)`
   - True
   - False

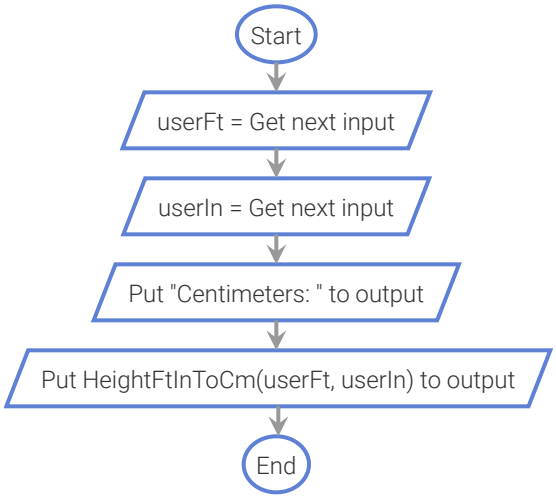
Mathematical functions

A function is commonly defined to compute a mathematical function involving several numerical parameters and returning a numerical result. The program below uses a function to convert a person's height in U.S. units (feet and inches) into total centimeters.

PARTICIPATION ACTIVITY | 4.2.4: Program with a function to convert height in feet/inches to centimeters.

Full screen

Main | HeightFtInToCm



Variables

0	userFt	integer
0	userIn	integer

Input

5 8

Output

ENTER EXECUTION | STEP | RUN | Execution speed: Medium

Human average height is increasing, attributed to better nutrition. Source: Wikipedia: Human height.

PARTICIPATION ACTIVITY 4.2.5: Mathematical functions.

Indicate which is a valid use of the HeightFtInToCm() function above. x is type float.

1) x = HeightFtInToCm(5, 0)
   - Valid
   - Not valid

2) x = 2 * (HeightFtInToCm(5, 0) + 1.0)
   - Valid
   - Not valid

3) x = (HeightFtInToCm(5, 0) + HeightFtInToCm(6, 1)) / 2.0
   - Valid
   - Not valid

4) Using the built-in function RaiseToPower, is the following valid? x = RaiseToPower(2.0, RaiseToPower(3.0, 2.0))
   - Valid
   - Not valid

Calling functions from functions

A function's statements may call other functions. In the example below, the PizzaCalories function calls the CircleArea function. (Note that the Main function itself is the first function called when a program executes, and calls other functions.)

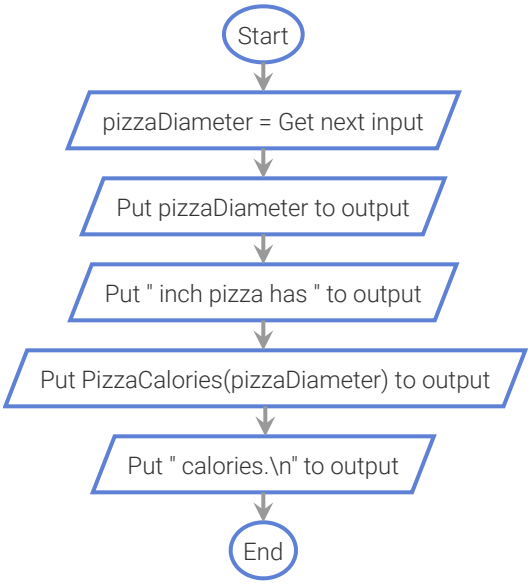
PARTICIPATION ACTIVITY 4.2.6: Functions calling functions.

Full screen

Main | PizzaCalories | CircleArea

Variables

0.0 pizzaDiameter float



Input

12

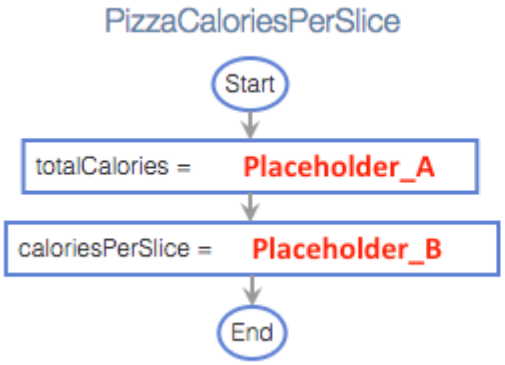
Output

–

NTER EXECUTION | STEP | RUN | Execution speed Medium

PARTICIPATION ACTIVITY 4.2.7: Functions calling functions.

Complete the PizzaCaloriesPerSlice() function to compute the calories for a single slice of pizza. A PizzaCalories() function returns a pizza's total calories given the pizza diameter passed as an argument. A PizzaSlices() function returns the number of slices in a pizza given the pizza diameter passed as an argument.



Parameter variables

0.0 pizzaDiameter float

Local variables

0.0 totalCalories float

Return variable

0.0 caloriesPerSlice float

1) Type the expression for Placeholder_A to compute the total calories for a pizza with diameter pizzaDiameter.

```
totalCalories =
```

Check Show answer

2) Type the expression for Placeholder_B to compute the calories per slice.

```
caloriesPerSlice =
```

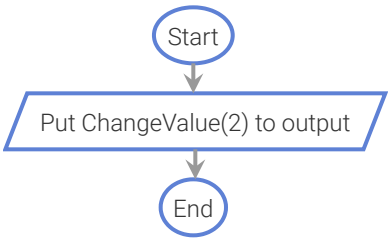
Check Show answer

CHALLENGE ACTIVITY 4.2.1: Enter the output of the returned value.

Start

Type the program's output

Main | ChangeValue



Variables

None

Output

8

1 2

Check Next

How was this section?

Provide feedback

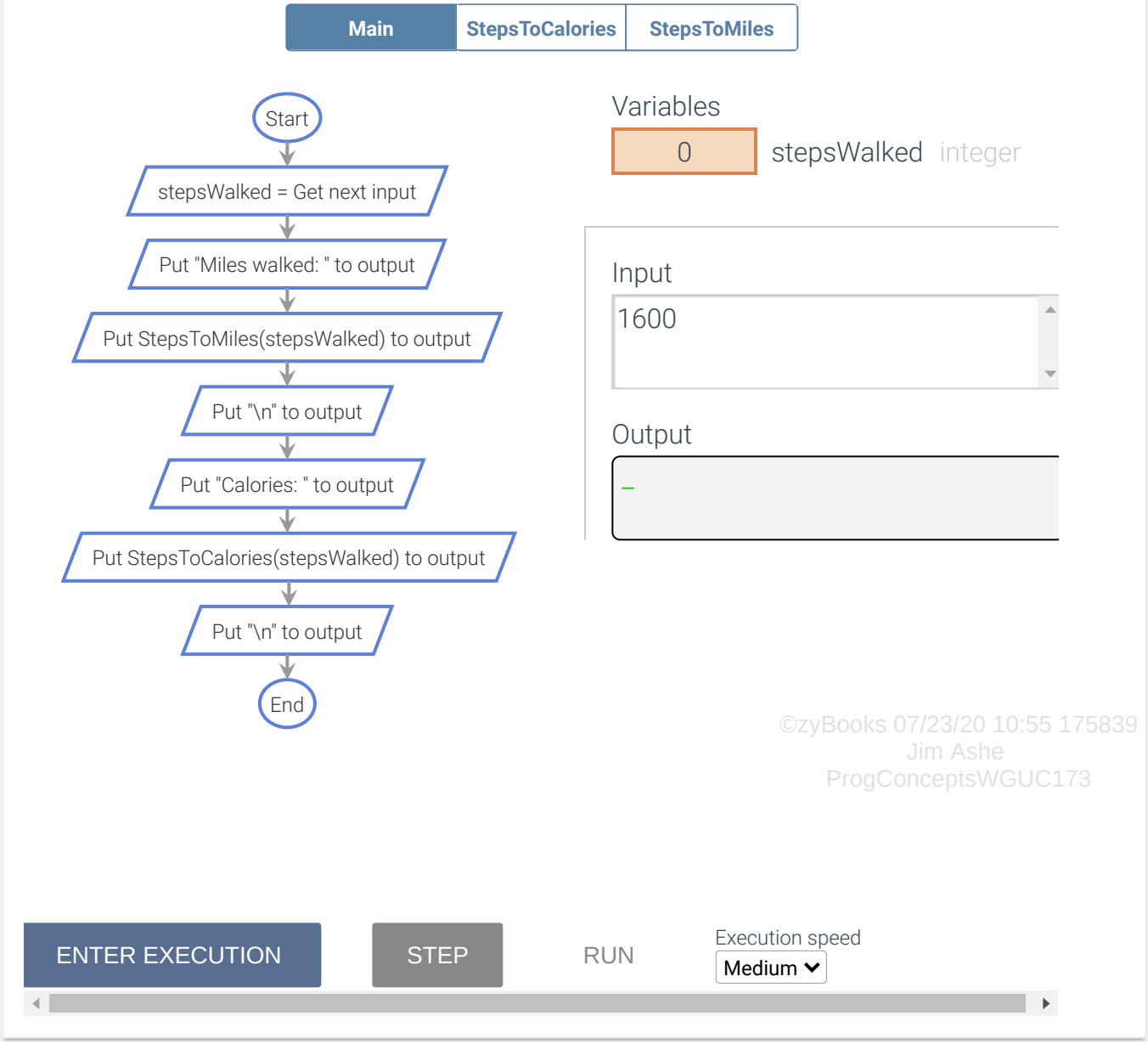
4.3 Reasons for defining functions

Improving program readability

Programs can become hard for humans to read and understand. Decomposing a program into functions can greatly aid program readability, helping yield an initially correct program, and easing future maintenance. Below, the Main function calls two other functions, rather than having too many details in Main itself, which keeps Main easy to read and understand. For larger programs, the effect is even greater.

PARTICIPATION ACTIVITY 4.3.1: With program functions: The Main function is easy to read and understand.

Full screen



PARTICIPATION ACTIVITY 4.3.2: Without program functions: The Main function is harder to read and understand. Full screen

Variables

0	stepsWalked	inte
0.0	feetPerStep	floa
0	feetPerMile	inte
0.0	stepsPerMinute	floa
0.0	calPerMinuteWalking	floa
0.0	minutesTotal	floa
0.0	caloriesTotal	floa
0.0	milesWalked	floa

Input

1600

Output

ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 4.3.3: Improved readability.

Consider the above examples. Include the Start and End statements.

1) In the example *without* functions, how many statements are in the Main function?

- 9
- 16

2) In the example *with* functions, how many statements are in the Main function?
   - 9
   - 16

3) Which has fewer *total* statements, the program with or without functions?
   - With
   - Without
   - Same

4) The program with functions called the functions directly in output statements. Did the program without functions directly put calculations in output statements?
   - No
   - Yes

Modular and incremental program development

Programmers commonly use functions to write programs modularly and incrementally.

- ***Modular development*** is the process of dividing a program into separate modules that can be developed and tested separately and then integrated into a single program.
- ***Incremental development*** is a process in which a programmer writes and tests a few statements, then writes and tests a small amount more (an incremental amount), and so on.
- A ***function stub*** is a function definition whose statements have not yet been written.

A programmer can use function stubs to capture the high-level behavior of the Main function and the required functions (or modules) before diving into details of each function, like planning a route for a road trip before starting to drive. A programmer can then incrementally develop and test each function independently.

PARTICIPATION ACTIVITY 4.3.4: Function stub used in incremental program development.

Animation content:

```
A zyFlowchart program follows:
Function float numMiles CcnvKilometersToMiles(float numKilometers)
   float milesPerKm

   milesPerKm = 0.621371
   numMiles = numKilometers * milesPerKm

Function nothing Main()
   float distKm
   float distMiles

   distKm = Get net input
   distMiles = ConvKilometersToMiles(distKm)
   Put "Miles driven: " to output
   Put distMiles to output

Variables in memory are as follows:
(empty) distKm: float
(empty) distMiles: float

Parameter variables in memory are as follows:
(empty) numKilometers: float

Return variable in memory are as follows:
(empty) numMiles: float

Local variables in memory are as follows:
(empty) milesPerKm: float
```


Animation captions:

1. Main captures the program's high-level behavior and calls a function stub. The program can be executed before writing that function.
2. The programmer then writes and tests the ConvKilometersToMiles function.

PARTICIPATION ACTIVITY 4.3.5: Incremental development.

1) Incremental development may involve more frequent execution and testing, but ultimately lead to faster development of a program.

True

False

2) The program above does not execute because ConvKilometersToMiles is a function stub.

True

False

3) A key benefit of function stubs is faster running programs.

True

False

4) Modular development means to divide a program into separate modules that can be developed and tested independently and then integrated into a single program.

True

False

Avoiding writing redundant statements

A function can be defined once, then called from multiple places in a program, thus avoiding redundant statements. Examples of such functions are math functions like AbsoluteValue that relieve a programmer from having to write several statements each time an absolute value needs to be computed.

The skill of decomposing a program's behavior into a good set of functions is a fundamental part of programming that helps characterize a good programmer. Each function should have easily-recognizable behavior, and the behavior of the Main function (and any function that calls other functions) should be easily understandable via the sequence of function calls.

A general guideline (especially for beginner programmers) is that a function's definition usually shouldn't have more than about 20 statements, although this guideline is not a strict rule.

PARTICIPATION ACTIVITY 4.3.6: Redundant statements can be replaced by multiple calls to one function.

Animation content:

```
A zyFlowchart program follows:
Function float circleArea CircleArea(float circleDiameter)
   float circleRadius

   piVal = 3.14159265
   circleRadius = circleDiameter / 2.0
   circleArea = piVal * circleRadius *  circleRadius

Function nothing Main()
   float pizzaDiameter1
   float pizzaDiameter2
   float totalPizzaArea
   float circleArea1
   float circleArea1

   pizzaDiameter1 = 12.0
   circleArea1 = CircleArea(pizzaDiameter1)
   pizzaDiameter2 = 14.0
   circleArea2 = CircleArea(pizzaDiameter2)
   totalPizzaArea = circleArea1 + circleArea2
   Put "A 12 and 14 inch pizza has " to output
   Put totalPizzaArea to output
   Put " inches squared combined. " to output


Variables in memory are as follows:
(empty) pizzaDiameter1: float
(empty) pizzaDiameter2: float
(empty) totalPizzaArea: float
(empty) circleArea1: float
(empty) circleArea1: float

Parameter variables in memory are as follows:
(empty) circleDiameter: float

Return variable in memory are as follows:
(empty) circleArea: float

Local variables in memory are as follows:
(empty) circleRadius: float
```


©zyBooks 07/23/20 10:55 175839
Jim Ashe
ProgConceptsWGUC173

Animation captions:

1. In this program, the statements for computing circle area appear twice. Such statements are redundant.
2. The redundant statements can be replaced by defining a CircleArea function.
3. Then main is simplified by calling the CircleArea function from two places.

PARTICIPATION ACTIVITY 4.3.7: Reasons for defining functions.

1) A key reason for creating functions is to help the Main function run faster.
   - True
   - False

2) Avoiding redundancy means to avoid calling a function from multiple places in a program.
   - True
   - False

3) If a function's statements are revised, all function calls will have to be modified too.
   - True
   - False

4) A benefit of functions is to increase redundant statements.
   - True
   - False

How was this section?

Provide feedback

4.4 Functions with branches/loops

Example: Auction website fee calculator

A function's statements may include branches, loops, and other statements. The following example uses a function to compute the amount that an online auction/sales website charges a customer who sells an item online.

PARTICIPATION ACTIVITY 4.4.1: Function example: Determining fees given an item selling price for an auction website.

Full screen

PARTICIPATION ACTIVITY 4.4.2: Analyzing the eBay fee calculator.

1) How many assignment statements for the variable `feeTotal` will

execute for the call EbayFee(20.0)?

Check Show answer

2) What does EbayFee(0.0) return (show your answer in the form #.##)?

Check Show answer

3) What does EbayFee(100.00) return (show your answer in the form #.##)?

Check Show answer

Example: Least-common multiple calculator

The following is another example with user-defined functions. The functions keep the Main function's behavior readable and understandable.

PARTICIPATION ACTIVITY 4.4.3: User-defined functions make the Main function easy to understand..

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 4.4.4: Analyzing the least common multiple program.

1) Other than the Main function, which user-defined function calls another user-defined function? Just write the function name.

Check Show answer

2) How many user-defined function calls exist in the program?

Check Show answer

3) How many function calls exist in the program?

Check Show answer

How was this section?

Provide feedback

4.5 Pseudocode: Functions

Function pseudocode basics

In pseudocode, a function definition begins by specifying the function's name and list of parameters. A function's parameters are listed in parentheses following the function's name, each parameter separated by a comma. A function definition with no parameters must still have the parentheses. The function's local variables and statements start on the following line and are indented.

PARTICIPATION ACTIVITY 4.5.1: Pseudocode for function with a parameter.

Animation captions:

1. Pseudocode for a function definition begins with the word "Function" followed by the function name.
2. The function's parameters are listed in parentheses after the function name. If a function has more than one parameter, each parameter is separated by a comma.
3. The function's local variables and statements start on the following line and are indented.

PARTICIPATION ACTIVITY 4.5.2: Parameters.

1) Complete the function to have a parameter named userAge of type integer.

```
Function PrintAge(
)
```

Check Show answer

2) Provide the first line of a function definition for a function named PrintMenu that has no parameters.

Check Show answer

3) Provide the first line of a function definition for a function named FindMax with two float parameters num1 and num2.

Check Show answer

4) Is the following a valid function definition beginning? Type Yes or No.

```
Function MyFct(integer userNum +
5)
```

Check Show answer

Function with a return value

If a function has a return value, the first line of the function definition defines the return variable's type and name. When the end of the function is reached, the value held in the return variable is returned.

Figure 4.5.1: Pseudocode for function with return.

```
Function HeightFtInToCm(integer heightFt, integer heightIn) returns float heightCm
   float cmPerInch
   integer inchesPerFeet
   integer totalInches

   cmPerInch = 2.54
   inchesPerFeet = 12

   // Total inches
   totalInches = (heightFt * inchesPerFeet) + heightIn
   // Convert inches to cm
   heightCm = totalInches * cmPerInch
```

PARTICIPATION ACTIVITY 4.5.3: Functions with return values.

1) The following is a valid function definition:

```
Function GetItem() returns float
newItem
```

- True
- False

2) The following is a valid function definition:

```
Function ConvertPosition(float
currPos) float newPos
```


- True
- False

3) The following is a valid function definition for a function that returns two items:

```
Function GetItems() returns
integer item1, integer item2
```

- True
- False

4) The following is a valid function definition:

```
Function PrintItem()
```

- True
- False

Explicitly returning a value

*In many programming languages, the return value is explicitly returned by a **return statement**. A return statement returns the specified value and immediately exits the function.*

```
Function CircleArea(float circleDiameter) returns float
   float circleRadius
   float piVal

   piVal = 3.14159265
   circleRadius = circleDiameter / 2.0
   circleArea = piVal * circleRadius * circleRadius

   return circleArea
```

How was this section?

Provide feedback

4.6 Functions summary

This chapter's key points included:

- A function is a named list of statements.
- A function definition consists of the new function's name and a block of statements. The function's name can be any valid identifier.
- A function call is an invocation of a function's name, causing the function's statements to execute.
- A program's execution begins with the Main function.

- A programmer can influence a function's behavior via an input. A parameter is a function input specified in a function definition. An argument is a value provided to a function's parameter during a function call.
- A parameter is like a variable declaration. Upon a call, the parameter's memory location is allocated, and the parameter is assigned with the argument's value.
- A function may return one value by assigning a return variable with the return value.
- A function call evaluates to the returned value. Thus, a function call often appears within an expression.
- Decomposing a program into functions can greatly aid program readability, helping yield an initially correct program, and easing future maintenance.
- Programmers commonly use functions to write programs modularly and incrementally.
- Modular development is the process of dividing a program into separate modules that can be developed and tested separately and then integrated into a single program.
- Incremental development is a process in which a programmer writes and tests a few statements, then writes and tests a small amount more (an incremental amount), and so on.
- A function can be defined once, then called from multiple places in a program, thus avoiding redundant statements.
- The skill of decomposing a program's behavior into a good set of functions is a fundamental part of programming that helps characterize a good programmer.
- Each function should have easily-recognizable behavior, and the behavior of the Main function (and any function that calls other functions) should be easily understandable via the sequence of function calls.
- A function's statements may include branches, loops, calls to other functions, and other statements.

PARTICIPATION ACTIVITY 4.6.1: Functions summary.

1) A function calculates the cost difference costDiff given two sample costs C1 and C2. What should be input to the function?
   - C1 only
   - C2 only
   - C1 and C2

2) A function calculates an average speed from the distance traveled and travel time. What should be output from the function?
   - Average speed
   - Distance traveled

- Travel time

3)

```
function G(integer x) returns
integer y
   y = (x + 3) * 2
```

What does G(7) evaluate to?

- 7
- 10
- 20

4)

```
Function Square(integer x) returns
integer y
   y = x * x

Function IncByTwo(integer x)
returns integer y
   y = x + 2
```

What does Square(IncByTwo(3)) evaluate to?

- 5
- 9
- 25

5)

```
Function PrintX()
   Put "X " to output

Function PrintY()
   Put "Y " to output
   PrintX()
```

What does PrintY() output?

- Y
- Y X
- X Y

6)

```
Function P(integer inNum) returns
integer outNum
   outNum = inNum * 3
```


What is inNum?

- A function parameter
- A function argument
- A function name

7)
```
Function CalcVal(integer num1,
integer num2) returns integer
resNum
   resNum = num1 + (num2 * 3)
```

What is resNum?

- A function parameter
- A function name
- A return variable

8)
```
Function CalcVal(integer num1,
integer num2) returns integer
resNum
   resNum = num1 + (num2 * 3)
```

For the function call CalcVal(4, 6), what is 4?

- A function parameter
- A function argument
- A return variable

9) What is the function argument in the following statements?

```
i = i + 1
yVal = SquareRoot(zVal * 2)
if (yVal < 4)
   wVal = wVal * 3
```

- zVal * 2
- SquareRoot
- wVal * 3

How was this section?

Provide feedback

©zyBooks 07/23/20 10:55 175839
Jim Ashe
ProgConceptsWGUC173

5.1 Algorithms

Problem solving and algorithms

Programs solve problems. Before writing a program, a programmer must first create an algorithm to correctly solve the given problem. An ***algorithm*** is a sequence of steps that solves a problem, generating correct output for any valid input values. Ensuring an algorithm's correctness is a key job for a programmer.

PARTICIPATION ACTIVITY 5.1.1: Before writing a program, a programmer must create a correct algorithm.

Animation content:

```
Problem: Computer the average of 5 fives, 2 1, 6, 2, 4

Algorithm 1 (Correct)
sum = 0
For each value
   sum = sum + value
Average = sum / 5

Result of algorithm 1 is 3

Algorithm 2 (Incorrect)
avg = 0
For each value
   avg = (avg + avlue) / 5

Result of algorithm 2 is 0.93024
```


Animation captions:

1. Problem: Compute the average of 5 values. A programmer may use an algorithm that initializes sum = 0, then adds each value to sum, then divides by 5.
2. A different algorithm initializes avg = 0, then for each value computes avg = (avg + value) / 5. But that algorithm is incorrect. avg = avg + (value / 5) would be correct.

PARTICIPATION

PARTICIPATION ACTIVITY

5.1.2: Is the algorithm correct?

Indicate whether the algorithm correctly solves the problem.

1) Problem: Kitchen is hot.
Algorithm:
(1) Open refrigerator door for 30 minutes to let cool air out.
(2) Close refrigerator door.

- No
- Yes

2) Problem: Traffic light won't turn green for bicyclist on the road.
Algorithm:
(1) Stand over the large loop normally under a car at the light.
(2) Jump up and down.
(3) Get back on bike and wait for green light.

- No
- Yes

3) Problem: Students in class don't know each other.
Algorithm:
(1) Have each student turn to left.
(2) Have student introduce themselves.
(3) Have each student face forward again.

- No
- Yes

4) Problem: A stack of 1000 name tags is unsorted but should be sorted.
Algorithm:
(1) Remove one tag, start a new stack.
(2) Remove another tag, place in sorted order in new stack.
(3) Repeat step 2 until no tags remain in original stack.

- No
- Yes

5) Problem: A stack of 1000 name tags is unsorted but should be sorted.
Algorithm:
(1) Start 26 new stacks for names starting with A, B, ..., Z.
(2) Remove one tag, place in appropriate new stack, in sorted order in that stack.
(3) Repeat step 2 until no tags remain in original stack.
(4) Stack the 26 stacks into one stack in sorted order.

- No
- Yes

6) Problem: Values for variables x and y should be swapped.
Algorithm:
(1) Copy y's value into x.
(2) Copy's x's value into y.

- No
- Yes

PARTICIPATION ACTIVITY 5.1.3: Ordering the steps of the algorithm.

Order the steps to correctly output the maximum of x and y.

max = x | **if (y > max) max = y** | **Put max to output**

(1)

(2)

(3)

Reset

PARTICIPATION ACTIVITY 5.1.4: Creating a correct algorithm.

Consider the problem of finding the maximum of five positive values, such as values 2 1 6 2 4. Indicate if the algorithm is correct, namely if max will end having the maximum value.

1)
```
max = 0
for each value
   if value > max
      max = value
   else
      max = 0
```

- Correct
- Incorrect

2)
```
max = 0
for each value
   if value > max
      max = value
```

- Correct
- Incorrect

3)
```
max = 0
for each value
   if max < value
      max = value
```

- Correct
- Incorrect

4) For a given algorithm, if max ends with 6 for the given data, the algorithm must be correct.

- True
- False

5) For a given algorithm, if max ends with a value other than 6, the algorithm must be incorrect.

- True
- False

Algorithm efficiency

Algorithms differ in various features. One feature is ***algorithm time efficiency***: The number of calculations required to solve a problem. For the same problem, some algorithms may be much more time efficient than others.

Simple programs are more likely to be correct, and correctness is always most important. Efficient algorithms tend to be less simple, so programmers should consider efficiency only when needed, like when dealing with very large data sets that result in slow program execution.

PARTICIPATION ACTIVITY | 5.1.5: Linear search and binary search are both correct algorithms, but binary search is more time efficient.

Animation content:

```
Problem: Determine if a value exists in an ascending list
Value to determine is 1983

Ascending list contains 1914, 1939, 1950, 1960, 1961, 1983, 1989,
1990, 1995

Linear search checks 1914, then 1939, then 1950, then 1960, then
1961, then 1984. A total of 6 checks was performed.
Binary search checks 1961, then 1989, then 1983. A total of 3 checks
was performed.
Thus, binary search was faster.
```

Animation captions:

1. Problem: Given an ascending list of values (start dates of U.S. wars), determine if a particular value exists (1983).
2. One algorithm is linear search: Starting with first item, check each item until found, or until reaching the list's end. Linear search may be slow.
3. Instead, binary search, checks the range's middle. If the value is greater, search is made on upper half. If less, on lower half. Search continues until found, or no range remains.
4. Both work, but binary search can be much faster. For 500 items, linear search may average 250 comparisons, but binary search will always use fewer than 10.

PARTICIPATION ACTIVITY | 5.1.6: Algorithm efficiency.

Consider linear search vs. binary search algorithms, on a list with 500 ascending values: 3, 7, 9, ..., 923, 925.

1) For linear search, about how many

comparisons are needed to determine if 924 exists?

- 2
- 250
- 500

2) For binary search, about how many comparisons are needed to determine if 924 exists?

- 2
- 10
- 500

3) For linear search, about how many comparisons are needed to determine if 5 exists?

- 2
- 250
- 500

4) For binary search, about how many comparisons are needed to determine if 5 exists?

- 2
- 10
- 250

5) Over large numbers of varied searches of values, linear search might average _____ comparisons, while binary search might average _____.

- 2, 10
- 500, 10
- 250, 10

PARTICIPATION ACTIVITY 5.1.7: Is the algorithm the most efficient?

Indicate whether the algorithm seems to be the most efficient algorithm to solve the problem.

1) Problem: Determine whether a number x is odd.
   Algorithm:
   (1) Count up from 1 by 2's: 1, 3, 5, ...
   (2) If x is encountered, x is odd.
   (3) If a number larger than x is encountered, x is not odd.

   - Yes
   - No

2) Problem: Given a list of numbers, determine whether the list's sum exceeds 1000.
   Algorithm:
   (1) Initialize sum = 0.
   (2) Start with the first number.
   (3) Add the current number to sum.
   (4) Repeat step 3 until all numbers have been added.
   (5) Compare sum with 1000.

   - Yes
   - No

How was this section?

Provide feedback

5.2 Algorithms summary

This chapter's key points included:

- A programmer must first create a correct algorithm: A sequence of steps to solve a problem.
- For large data, also relevant is an algorithm's time efficiency: The number of calculations needed to solve a problem.

PARTICIPATION ACTIVITY 5.2.1: Algorithms.

Note: An algorithm is "correct" if the algorithm yields correct output for all valid input values.

1) A programmer creates an algorithm to determine whether a number is a prime number. The programmer tests the algorithm for an input of 7, and the algorithm yields the correct output. Is the algorithm correct?
   - Yes
   - No
   - Unknown

2) A programmer creates an algorithm to determine whether a number is a prime number. The programmer tests the algorithm for an input of 13, and the algorithm yields an incorrect output. Might the algorithm still be correct?
   - Yes
   - No
   - Unknown

3) Is the following an algorithm?
   The pool's water level is low. Water should be added.
   - Yes
   - No

4) Is the following an algorithm?
   Measure the pool's water level. If low, add water. Stop when the level is sufficient.
   - Yes
   - No

5) Two algorithms each correctly sort a list of numbers. Algorithm 1 sorts 1000 numbers in 5 sec. Algorithm 2's code is more complex, but sorts

1000 numbers in 1 sec. Which is true?

- Algorithm 2 is correct but Algorithm 1 is incorrect.
- Algorithm 2 is more time-efficient.
- Algorithms 1 and 2 are both incorrect.

How was this section?

Provide feedback

©zyBooks 07/23/20 10:58 175839
Jim Ashe
ProgConceptsWGUC173

6.1 Software design processes

System development life cycle

A 4-bedroom house can be built using different arrangements of phases. For one house, first a 4-bedroom blueprint might be created, and then the 4-bedroom house built per that blueprint. For another house, first a 1-bedroom blueprint might be created, then that 1-bedroom house built (and moved into), then blueprints for more rooms created, and then those rooms built.

Similarly, a program can be built using different arrangements of phases, comprising the ***systems development life cycle*** or ***SDLC***.

- The ***analysis phase*** defines a program's goals.
- The ***design phase*** defines specifics of how to build a program.
- The ***implementation phase*** involves writing the program.
- The ***testing phase*** checks that the programs correctly meets the goals.

PARTICIPATION ACTIVITY 6.1.1: SDLC and phases.

Match the phases of the software project with the phase name.

A programmer writes each function using statements.

Each calculation will be a function. The user types a letter to invoke each calculation.

The program should support mean, median, mode, max, and min calculations.

A programmer checks that each function works.

	Analysis
	Design
	Implementation

Testing

Reset

Waterfall versus agile approach

A program can be built by carrying out the SDLC phases in sequence, known as the ***waterfall approach***. The term waterfall is used because, just like a boat going down river doesn't come back, no earlier phase is come back to. In contrast, a program can be built by doing small amounts of each SDLC phases in sequence, and then repeating, known as the ***agile approach*** (or ***spiral approach***).

PARTICIPATION ACTIVITY 6.1.2: Systems development life cycle (SDLC) phases include analysis, design, implementation and testing, done using a waterfall or spiral process.

Animation content:

```
Waterfall method:
Analysis, followed by Design, followed by Implementation, followed
by Testing

Spiral method:
Analysis, followed by Design, followed by Implementation, followed
by Testing, followed by Analysis, followed by Design, followed by
Implementation, followed by Testing
```

Animation captions:

1. Analysis determines the goals for a program. Design determines the specifics of how to build the program.
2. Implementation builds the program according to the design. Testing checks that the program meets the goals.
3. The waterfall approach executes the SDLC phases sequentially. Each phase begins after the previous phase is done. No phase repeats.
4. The spiral (or agile) approach goes through the phases repeatedly, each time adding more to the program.

PARTICIPATION ACTIVITY 6.1.3: Waterfall and agile approaches.

1) A 4-bedroom house's blueprint is created, then the 4-bedroom house is built. What SDLC approach is represented?
   - Waterfall
   - Agile

2) A 1-bedroom house's blueprint is created, then the 1-bedroom house is built (and moved into). Later, the blueprint is extended with 2 more bedrooms, and those are built. Later, 1 more bedroom is added to the blueprint, and then built.
   - Waterfall
   - Agile

3) Defining and building a house a few bedrooms at a time may have an advantage of letting people move in sooner.
   - True
   - False

4) Programmers analyze smartphone users and decide that a new game is needed that launches angry birds at objects. The programmers decide to build a game with 1 kind of bird and 1 difficulty level. What phases have been carried out?
   - Analysis and design
   - Implementation and testing

5) After deciding to build an angry birds game with 1 bird and 1 level, programmers write software for the game, test the game, and release the game to 100 users. After seeing user feedback, the programmers decide to extend the game to have 2 kinds of birds and 3 levels. What

SDLC approach are the programmers using?

- Waterfall
- Agile

6) Which approach is more responsive to user feedback about a program?

- Waterfall
- Agile

7) In which approach would carrying out one phase, like implementation, likely take longer?

- Waterfall
- Agile

SDLC case study

A programmer is tasked with writing a web program that helps people choose among home loans. The programmer:

- Talks with 30 potential customers, learning most want to choose between different 30-year fixed rate loans, but some want to also consider 20-year and 15-year loans, and variable rate loans too. The programmer decides to support all those kinds of loans. Time: 3 weeks.
- The programmer decides to write a program running on a remote computer that generates a web page, with fields for a user to enter loan amount and years, and implementing each loan type as a function. Time: 1 week.
- The programmer implements the program for every loan type. Time: 3 weeks.
- The programmer tests the program thoroughly and fixes bugs. Time: 1 week.

The programmer releases the program to customers. Immediately, customers request a graphical comparison of the loan payment schedules, which would be easier to do using a different programming language running in the web browser. Users also rarely use the 20-year, 15-year, and variable rate options, and many state that having all those options is confusing. These changes will be quite time consuming for the programmer, who has already invested much time, the programmer leaves the program as is. The program is not very popular.

PARTICIPATION ACTIVITY 6.1.4: Waterfall and agile approaches.

Consider the example above.

1) The programmer used what SDLC approach?
   - Waterfall
   - Agile
   - Spiral

2) The programmer spent about 2 months, yet created a program that wasn't popular. The programmer might have been better off talking to fewer customers and then building a program that supported _____ loans.
   - more
   - fewer

3) If the programmer initially created a program supporting fewer loans, the programmer would have more quickly learned that people wanted graphical comparisons. The programmer could have then designed, implemented, and tested a graphical display. That approach is known as _____.
   - waterfall
   - agile
   - wasteful

How was this section?

Provide feedback

6.2 Software design processes summary

This chapter's key points included:

- Programs may be developed in various phases known as the system development life cycle (SDLC).

- Programs may be developed in phases known as the system development life cycle (SDLC), wherein analysis defines a program goals, design defines specifics of how to build a program, implementation writes the program, and testing checks the program.
- A waterfall approach does the phases in sequence once, while an agile approach (aka spiral approach) does smaller amounts of each phase and repeats.

PARTICIPATION ACTIVITY 6.2.1: System design life cycle.

1) What SDLC phase emphasizes defining a program's goals?
   - Analysis
   - Design
   - Implementation

2) In an agile approach, what is the maximum number of times that the implementation phase is carried out?
   - 0
   - 1
   - No max

How was this section?

Provide feedback

7.1 Objects: Introduction

Grouping things into objects

The physical world is made up of material items like wood, metal, plastic, fabric, etc. To keep the world understandable, people deal with higher-level objects, like chairs, tables, and TV's. Those objects are groupings of the lower-level items.

Likewise, a program is made up of items like variables and functions. To keep programs understandable, programmers often deal with higher-level groupings of those items known as objects. In programming, an ***object*** is a grouping of data (variables) and operations that can be performed on that data (functions).

PARTICIPATION ACTIVITY 7.1.1: The world is viewed not as materials, but rather as objects.

Animation captions:

1. The world consists of items like, wood, metal, fabric, etc.
2. But people think in terms of higher-level objects, like chairs, couches, and drawers.
3. In fact, people think mostly of the operations that can be done with the object. For a drawer, operations are put stuff in, or take stuff out.

PARTICIPATION ACTIVITY 7.1.2: Programs commonly are not viewed as variables and functions/methods, but rather as objects.

Animation captions:

1. A program consists of variables and functions/methods. But programmers may prefer to think of higher-level objects like Restaurants and Hotels.
2. In fact, programmers think mostly of the operations that can be done with the object, like setting main info, or adding a review.

PARTICIPATION ACTIVITY 7.1.3: Objects.

Some of the variables and functions for a used-car inventory program are to be grouped into an object type named CarOnLot. Select True if the item should become part of the CarOnLot object type, and False otherwise.

1) int carStickerPrice;

- True
- False

2) double todaysTemperature;

- True
- False

3) int daysOnLot;

- True
- False

4) int origPurchasePrice;

- True
- False

5) int numSalespeople;

- True
- False

6) GetDaysOnLot()

- True
- False

7) DecreaseStickerPrice()

- True
- False

8) DetermineTopSalesperson()

- True
- False

Abstraction / Information hiding

Abstraction means to have a user interact with an item at a high-level, with lower-level internal details hidden from the user (aka ***information hiding*** or ***encapsulation***). Ex: An oven supports an

abstraction of a food compartment and a knob to control heat. An oven's user need not interact with internal parts of an oven.

Objects strongly support abstraction, hiding entire groups of functions and variables, exposing only certain functions to a user.

An ***abstract data type*** (***ADT***) is a data type whose creation and update are constrained to specific well-defined operations. A class can be used to implement an ADT.

PARTICIPATION ACTIVITY 7.1.4: Objects strongly support abstraction / information hiding.

Animation captions:

1. Abstractions simplifies our world. An oven is viewed as having a compartment for food, and a knob that can be turned to heat the food.
2. People need not be concerned with an oven's internal workings. Ex: People don't reach inside trying to adjust the flame.
3. Similarly, an object has operations that a user can apply. The object's internal data, and possibly other operations, are hidden from the user.

PARTICIPATION ACTIVITY 7.1.5: Abstraction / information hiding.

1) A car presents an abstraction to a user, including a steering wheel, gas pedal, and brake.
   - ○ True
   - ○ False

2) A refrigerator presents an abstraction to a user, including freon gas, a compressor, and a fan.
   - ○ True
   - ○ False

3) A software object is created for a soccer team. A reasonable abstraction allows setting the team's name, adding or deleting players, and printing a roster.
   - ○ True
   - ○ False

4) A software object is created for a university class. A reasonable abstraction allows viewing and modifying variables for the teacher's name, and variables implementing a list of every student's name as well.

- True
- False

How was this section?

Provide feedback

7.2 UML

Universal Modeling Language

The ***Universal Modeling Language*** (***UML***) is a modeling language for software design that uses different types of diagrams to visualize the structure and behavior of programs. UML consists of several structural and behavioral diagrams. A ***structural diagram*** visualizes static elements of software, such as the types of variables and functions used in the program. A ***behavioral diagram*** visualizes dynamic behavior of software, such as the flow of an algorithm. For example, A UML ***activity diagram*** is a flowchart, similar to zyFlowchart, used to describe the flow of an activity or set of activities.

PARTICIPATION ACTIVITY 7.2.1: UML has various diagrams, like activity, use case, class, and sequence diagrams.

Animation captions:

1. UML has several kinds of diagrams. An activity diagram is like a flowchart, describing at a high-level the sequence of computations.
2. A use case diagram shows various things a user might do. A user might enter data, compute an absolute value, or compute a mean.
3. A class diagram shows a program's parts (classes). A DataSet class might compute statistics on data. A DataVisualizer does that plus draws bar charts and histograms.
4. A sequence diagram shows interaction between software components and indicates order of events. Communication between client and server for a web search might be

shown with a sequence diagram.

PARTICIPATION ACTIVITY 7.2.2: Behavioral and structural UML.

For each task, select whether a UML behavioral diagram or a UML structural diagram should be used to describe an aspect of software.

1) List of components necessary to build an online shopping system.
   - ○ Behavioral
   - ○ Structural

2) Overview showing how a customer uses a particular software product.
   - ○ Behavioral
   - ○ Structural

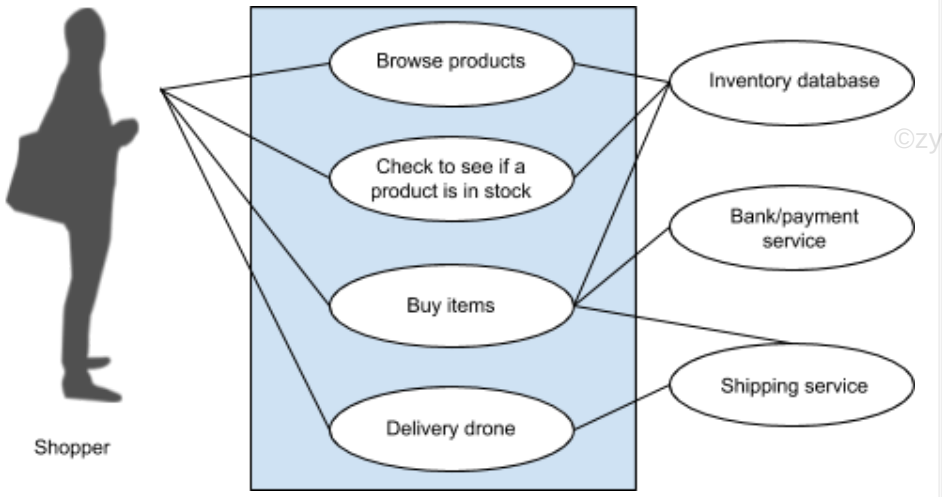
3) Flowchart illustrating a program's logic, including operations, branches, and loops, for converting a multiple temperature measurements from C to F.
   - ○ Behavioral
   - ○ Structural

Use case diagram

A UML ***use case diagram*** is a behavioral diagram used to visually model how a user interacts with a software program. Actions from users and the accompanying actions in software, as well as connections between different components of the software, are illustrated in a use case diagram. Use case diagrams are often used to specify behavioral requirements of programs.

Ex: The use case diagram below shows components of an online shopping system and how the customer, as well as how external services, connect to system components.

Figure 7.2.1: UML use case diagram for online shopping system.



PARTICIPATION ACTIVITY 7.2.3: Use case diagram.

1) A use case diagram contains program code.
   - True
   - False

2) A use case diagram represents what the user will see on screen when using the software.
   - True
   - False

3) Use case diagrams indicate where different components in software connect to each other.
   - True
   - False

4) A use case diagram always indicates order of events.
   - True
   - False

5) A use case diagram is a structural

UML diagram.

- True
- False

Class diagram

A UML ***class diagram*** is a structural diagram that can be used to visually model the classes of a computer program, including data members and functions. A ***class*** is a code blueprint for creating an object that is composed of data members and functions that operate on those data members.

PARTICIPATION ACTIVITY 7.2.4: UML class diagrams.

Animation captions:

1. A class diagram depicts a class' name, data members, and functions. The "Car" class consists of two string member variables "make" and "model".
2. The CarForSale class has a float member variable salePrice and an integer daysOnLot.
3. A car for sale is still a car, so the CarForSale class can inherit the make and model members from the Class, using a programming feature called inheritance.
4. The UsedCarLot class has a cars member that store a list of Cars and a sellCar member function to sell a car on the lot.
5. The unfilled diamond indicates the UsedCarLot class "has" CarForSale items, which are elements of the class' cars list.

PARTICIPATION ACTIVITY 7.2.5: UML class diagrams.

1) UML class diagrams can indicate member variable data types.
   - True
   - False
2) UML class diagrams can depict one class inheriting data members and functions of another class.
   - True
   - False
3) UML class diagrams can depict high-level business use cases.

- True
- False

4) A UML class diagram can depict if a class has data members of another class type.
   - True
   - False

Sequence diagram

A UML ***sequence diagram*** is a behavioral diagram that shows interaction between software components and indicates the order of events. A UML sequence diagram is commonly used to illustrate the sequence of events needed to handle a particular scenario in software.

PARTICIPATION ACTIVITY 7.2.6: UML sequence diagram.

Animation captions:

1. A sequence diagram can be used to illustrate the sequence of events for a web search.
2. Horizontal arrows are used to indicate communication between the computer and the web server.
3. Vertical dashed lines represent the lifelines of the computer and server.
4. Horizontal dashed lines represent responses from the server.
5. The diagram indicates that the computer sends the search term and gets 50 results back. Later, the computer requests the next 50 results, which are sent back by the server.

PARTICIPATION ACTIVITY 7.2.7: Sequence diagram.

1) A sequence diagram is a structural UML diagram.
   - True
   - False

2) A sequence diagram always indicates order of events.
   - True
   - False

3) Sequence diagrams contain

program code.

- True
- False

UML and SDLC

UML is commonly used in various phases of the systems development life cycle (SDLC), whether using a waterfall or agile approach.

PARTICIPATION ACTIVITY 7.2.8: UML diagrams used in SDLC phases.

Animation content:

undefined

Animation captions:

1. During SDLC's analysis phase (define program goals), a use case diagram can indicate what goals (use cases) the application should carry out.
2. During design (specifics of how to build), a class diagram can lay out how to group data and functions.
3. During implementation (writing the program), an activity diagram can describe the program's instructions, branches, loops, etc.
4. Finally, during testing (check that program meets goals), a sequence diagram can ensure the program generates outputs for given inputs, in the expected order.

PARTICIPATION ACTIVITY 7.2.9: UML and SDLC.

Indicate which UML diagram is most directly useful for the given SDLC phase. Whether a waterfall or agile approach is followed is not relevant, since UML diagrams can be used for the phases in either approach.

Use case **Class diagram** **Activity diagram** **Sequence diagram**

	Analysis
	Design

	Implementation
	Testing

Reset

How was this section?

Provide feedback

7.5 UML summary

This chapter's key points included:

- UML uses different kinds of diagrams to visualize programs. A structural diagram shows static items like variables and functions. A behavioral diagram shows dynamic behavior like flow.
- A UML use case diagram is behavioral and shows how a user interacts with a program. A class diagram is structural and shows a program's classes. A sequence diagram is behavioral and shows interactions and event orderings.

PARTICIPATION ACTIVITY 7.5.1: UML.

1) Which diagram is most likely to be part of the analysis phase of the SDLC, which defines a program's goals?
   - Use case diagram
   - Activity diagram
   - Flow chart

2) Which diagram is most likely to be part of the design phase of the SDLC, which describes how a program will be built?
   - Use case diagram
   - Class diagram
   -

Activity diagram

3) Which diagram is most likely to be part of the testing phase of the SDLC?

- Class diagram
- Sequence diagram

How was this section?

Provide feedback

8.1 Language survey

Compiled vs. interpreted languages

Many kinds of programming languages have evolved, serving different purposes. One kind is a compiled language. A program written in a ***compiled language*** is first converted by a tool (***compiler***) into machine code, which can run on a particular machine. Examples include C, C++, Java, and C#.

In contrast, an ***interpreted language*** (aka ***scripting language***) is a language that is run one statement at a time by another program called an ***interpreter***. Examples include Python, Javascript, and MATLAB.

Interpreted languages tend to be easier for new programmers, who don't have to run a compiler. Also, an interpreted language's program can run on any machine that has an interpreter. But compiled languages run faster.

PARTICIPATION ACTIVITY 8.1.1: Compiled and interpreted languages.

Animation captions:

1. A programmer writes a high level program, known as source code.
2. For a compiled language, the programmer runs a compiler, which converts the high level program into an executable program.
3. Users can then run the executable.
4. For an interpreted language, the programmer still writes a high level program. But then a user runs an interpreter to run that program.
5. Interpreted languages are easier on the programmer and can run on any machine that has the interpreter, but may run more slowly.

PARTICIPATION ACTIVITY 8.1.2: A sample program in C++, a compiled language.

Below is a program in C++, which is a compiled language. The program has an error. Normally a programmer first compiles a program, then runs the resulting executable. Below, the "Run" button does both.

- Press Run and notice the error reported by the compiler.
- Fix the error by typing a ; at line 6's end.
- Press Run again to see the program execute.

Load default template...

Run

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

int main() {
   cout << "This is a sample program." << endl;
   cout << "It just outputs text." << endl
   cout << "Hello there, and goodbye." << endl;
   return 0;
}
```


PARTICIPATION ACTIVITY 8.1.3: Compiled vs. interpreted languages.

1) Which kind of language is first converted to an executable, then run?
   - Compiled
   - Interpreted

2) Which kind of language can be run right away, without converting to an executable first?
   - Compiled
   - Interpreted

3) Which kind of language is more portable to different machines?
   - Compiled
   - Interpreted

4) Which kind of language usually has faster execution?
   - Compiled
   - Interpreted

5) C, C++, and Java are what kind of

language?

- Compiled
- Interpreted

6) Python, Javascript, and MATLAB are each what kind of language?

- Compiled
- Interpreted

Statically vs dynamically typed languages

Most compiled languages are ***statically typed***, meaning each variable's type must be declared and cannot change while the program runs. ("Static" means unchanging).C, C++, and Java are popular examples. In contrast, many interpreted languages are ***dynamically typed***, meaning a variable's type may change while a program executes, usually based on what is assigned to the variable. ("Dynamic" means changing). Python is a popular example.

Statically-typed languages are often considered safer due to reporting errors during compilation (like if assigning an integer with a string), while dynamically-typed languages are considered easier to use when types need to change, requiring less converting or fewer variables. Much debate exists.

PARTICIPATION ACTIVITY 8.1.4: Python is dynamically typed, meaning a variable's type can change while running.

Press Run to view the below program's output.

- The program first assigns x with a string, thus creating a variable x of string type.
- The program then assigns variable x with an integer, which changes x's type to an integer.
- The program then assigns x with x * 1.5. Because the result is a float, x's type changes to a float.

Load default template...

Run

```
1
2 x = "Hello"
3 print(x)
4 x = 5
5 print(x)
6 x = x * 1.5
7 print(x)
8
```

PARTICIPATION ACTIVITY 8.1.5: C++ is statically typed.

The program below declares an integer variable named x.

- The program first assigns x with string "Hello". No conversion exists from string to integer. Press Run and note the program fails to compile. Remove that line (or comment the line out by prepending //) and press Run again.
- The program assigns x with 5.
- The program then assigns x with x * 1.5. The result should be 7.5 (a float type), but because x is an integer, a conversion occurs: The fraction is ignored, and x gets 7.

Load default template... Run

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

int main() {
   int x;
   x = "Hello";
   x = 5;
   cout << x << endl;
   x = x * 1.5;
   cout << x << endl;
   return 0;
}
```

PARTICIPATION ACTIVITY 8.1.6: Statically vs. dynamically typed languages.

1) In a statically-typed language, a variable x is declared as an integer type. Later, x is assigned with 3.14, after which x's type is _____.

   - ○ an integer

- a float

2) In a dynamically-typed language, a variable x is declared as an integer type. Later, x is assigned with 3.14, after which x's type is _____.
   - an integer
   - a float

3) In a statically-typed language, if a programmer assigns integer variable numPeople with "Hello there", an error ___ usually be reported.
   - will
   - will not

4) In a dynamically-typed language, if a programmer assigns integer variable userInput with "Hello there", an error ___ usually be reported.
   - will
   - will not

5) Dynamically-typed languages are better than statically-typed languages.
   - True
   - False

Object-oriented languages

A large program thought of as thousands of variables and functions is hard to understand. A higher level approach is needed to organize a program in a more understandable way.

In the physical world, people are surrounded by basic items made from wood, metal, plastic, etc. But to keep the world understandable, we think at a higher level, in terms of *objects* like an oven. The oven allows us to perform a few specific operations, like put an item in the oven, or set the temperature.

Thinking in terms of objects can be powerful when designing programs. Suppose a program should record time and distance for various runners, such as a runner ran 1.5 miles in 500 seconds, and should compute speed. A programmer might think of an "object" called RunnerInfo. The RunnerInfo object supports

operations like setting distance, setting time, and computing speed. In a program, an ***object*** consists of some internal data items plus operations that can be performed on that data.

PARTICIPATION ACTIVITY | 8.1.7: Grouping variables and functions into objects keeps programs understandable.

Animation content:

On left there is no grouping of variables or functions:

```
double distRun;
int timeRun;
double GetSpeed()
void PrintRunnerStats()
int numSpectators;
int ticketPriceNormal;
int numStudents;
int ticketPriceStudent;
int CalculateRevenue()
```

On right there is a grouping of variables or functions to keep programs more understandable:

```
RunnerInfo contains the following variables and functions:
double distRun;
int timeRun;
double GetSpeed()
void PrintRunnerStats()

CrowdInfo contains the following variables and functions:
int numSpectators;
int ticketPriceNormal;
int numStudents;
int ticketPriceStudent;
int CalculateRevenue()
```


Animation captions:

1. A program with many variables and functions can be hard to understand
2. Grouping related items into objects keeps programs understandable

Creating a program as a collection of objects can lead to a more understandable, manageable, and properly-executing program. An ***object-oriented language*** supports decomposing a program into objects. C++, Java, Python, and C# provide extensive object-oriented support. In fact, the "++" in C++ suggests C++ is better than C, due in large part to supporting objects. C does not. MATLAB and Javascript provide some support.

PARTICIPATION ACTIVITY 8.1.8: Objects.

Some of the variables and functions for a used-car inventory program are to be grouped into an object type named CarOnLot. Select True if the item should become part of the CarOnLot object type, and False otherwise.

1) integer carStickerPrice
   - True
   - False

2) float todaysTemperature
   - True
   - False

3) integer daysOnLot;
   - True
   - False

4) integer origPurchasePrice
   - True
   - False

5) integer numSalespeople
   - True
   - False

6) IncrementCarDaysOnLot()
   - True
   - False

7) DecreaseStickerPrice()
   - True
   -

False

8) DetermineTopSalesperson()

- True
- False

PARTICIPATION ACTIVITY 8.1.9: Object-oriented languages.

1) A large program will be written for a school system. A school has people, who have names, ages, etc. Some people have report cards, which has courses, grades, teachers, etc. A course has students, a teacher, a room, etc. An object-oriented language is likely preferred.

- True
- False

2) A large program will be written for data analysis. Given a file of data, a user can apply different operations, like compute mean, compute median, compute max, etc. An object-oriented language is likely preferred.

- True
- False

3) Which language has extensive support for objects?

- C
- C++

Markup languages

A ***markup language*** allows a developer to describe a document's content, desired formatting, or other features. A popular markup language is HTML. ***HTML*** (hyper-text markup language) allows a developer to describe the text, links, images, and other features of a web page. While some people refer to "HTML programming", a markup language is not executed statement-by-

statement like a programming language, and an HTML file is not a program. Rather, a web browser reads an HTML file and renders the corresponding web page. An HTML file surround text by different *tags* to yield different formatting.

PARTICIPATION ACTIVITY 8.1.10: HTML viewer.

The HTML on the left is rendered on the right.

- Notice the text surround by h1 tags. Add some text using h2 tags, render, and notice the difference between h1 and h2 tags.
- Try underlining some text by surrounding that text by <u> and </u> tags.

Reset

Type HTML below

```
<!DOCTYPE html>
<html>


<head>
<title>My Page's Title</title>
</head>

<body>
 <h1> Is Pluto a Planet?  </h1>
<p> When I was young,  <i> Pluto </i>
was a planet. Then later, it wasn't.
</p>
<p>  Now, I hear Pluto may be a planet
again. Well, <em> is it or isn't it??
</em>
 </p>
```

Rendered HTML

Is Pluto a Pla

When I was young, *Pluto* was

Now, I hear Pluto may be a pl

Render HTML

PARTICIPATION ACTIVITY 8.1.11: Markup languages: HTML.

1) Putting <h1> and </h1> around text causes a web browser to render that text using large bold font.
   - ○ True
   - ○ False

2) A web browser executes an HTML program one statement at a time.

- True
- False

3) An HTML file is typically compiled into an executable file, which can then be run.

- True
- False

Summary

Numerous kinds of languages exist, each having advantages and disadvantages, and serving different purposes.

PARTICIPATION ACTIVITY 8.1.12: Kinds of languages.

Match the kind of language with the description. Note that the kinds listed are not exclusive; for example, an object-oriented language may be statically-typed or dynamically-typed.

Statically-typed **Compiled** **Dynamically-typed** **Markup** **Interpreted**

Object-oriented

	A language that supports decomposing a large program into a set of items.
	A language whose variable types do not change during execution.
	A language whose variable types may change during execution.
	A programming language that is first converted to machine code,

	which can then only run on a particular machine type.
	A programming language that requires no compilation, and instead can be directly run by another program.
	A language that puts tags around text to indicate formatting and other features.

Reset

How was this section?

Provide feedback

8.2 Libraries

Library basics

A ***library*** is a set of pre-written functions (and other items) that carry out common tasks, that a programmer can use to improve productivity. After including a library with a program, a programmer can make use of the library's functions. A library's functions typically all relate to the same purpose, like computing statistics, or like displaying graphics. A programmer may include several libraries in a single program.

PARTICIPATION ACTIVITY 8.2.1: Using libraries can improve programmer productivity.

Animation content:


```
Left is a StatisticLibrary with the following functions:

ComputeMean(...)
...
ComputeMedian(...)
```

```
...
ComputeMax(...)
...

Right shows a new program that uses the StatisticLibrary:
Include StatisticsLibrary

// userList is 2, 8, 3, 7
mean = ComputeMean(userList)
max = ComputeMax(userList)
```


Animation captions:

1. A library is a set of pre-written functions (and other items). This library has functions to compute statistics, like mean, median, and max.
2. After including the library, a programmer can immediately use the library's functions, thus improving productivity.

PARTICIPATION ACTIVITY 8.2.2: Libraries.

1) To use a library, a programmer must include the library, and then write the functions for the library.
   - True
   - False

2) A key advantage of using a library is improved programmer productivity.
   - True
   - False

3) A programmer can only use one library in a program.
   - True
   - False

4) A typical library consists of one function.
   - True
   - False

How was this section?

Provide feedback

8.3 Language survey / libraries summary

This chapter's key points included:

- Many kinds of languages exist. Compiled languages are first converted to machine code, while interpreted languages instead run on an interpreter. Statically typed languages require a programmer to declare a variable's type, which cannot change, while dynamically typed languages let the type change as a program runs. Object-oriented languages provide substantial support for decomposing a program into objects. Markup languages don't execute, but describe formatting and other features (like HTML, the language of web pages).
- Programmers use libraries to improve productivity, by making use of pre-written functions.

PARTICIPATION ACTIVITY 8.3.1: Language types, and libraries.

1) Which language(s) is/are statically typed rather than dynamically typed?
   - C, C++, and Java
   - Python

2) Which language(s) is/are compiled rather than interpreted?
   - C, C++, and Java
   - Python

3) Which language is NOT considered object-oriented?
   - C
   - C++
   - Java

4) What kind of language is HTML, the language of web pages?
   - Object-oriented
   -

Statically-typed

Markup

5) What is typically found in a library?

A user's main program

Pre-written functions

A list of scalar variables

How was this section?

Provide feedback