CSC 130: C Programming

I/O
Data types
Memory
Operators: arithmetic and relational

Reynolds
COMMUNITY COLLEGE

C Programming

- printf() and scanf() functions
- Fundamental data types in C
- Computer memory concepts
- C arithmetic operators
- C decision making statements

- Internet C reference that appears to be accurate
- http://www.tutorialspoint.com/cprogramming/c_basic_syntax.htm

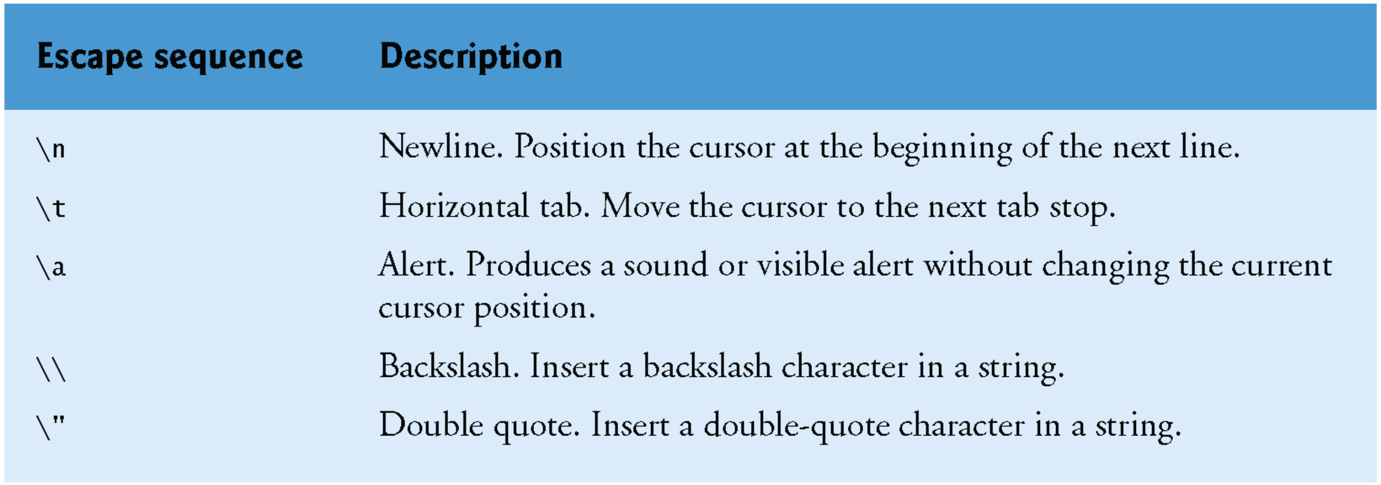
Read input and write output

- I/O functions are in the Standard C Library
- We have used printf() to output a string
- Input to a running program is originated at its command prompt by the scanf() function
- Input values are stored in variables in memory
  - Memory blocks are given variable names
  - Memory block size is determined by its data type
  - Memory block content is set by an assignment

printf()

- **int printf(const char *format, ...)**
  - A string literal may be written as output
  - A format string is used to format variables
- Writes to the standard output stream **stdout**
- Output may be formatted using a format string
  - printf(“hello world is a string”);
  - printf(“this is an integer %d”, 42);
- Output may include escape characters

Escape characters used by printf()



Escape sequence	Description
`\n`	Newline. Position the cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the cursor to the next tab stop.
`\a`	Alert. Produces a sound or visible alert without changing the current cursor position.
`\\`	Backslash. Insert a backslash character in a string.
`\"`	Double quote. Insert a double-quote character in a string.

Fig. 2.2 | Some common escape sequences .

scanf()

- **int scanf(const char *format, ...)**
- Reads from the standard input stream **stdin**
- Data type determined by a **format** pattern
- Returns number of items read

- For example:
- scanf(**"%d"**, **&integer1**); **/* read integer */**

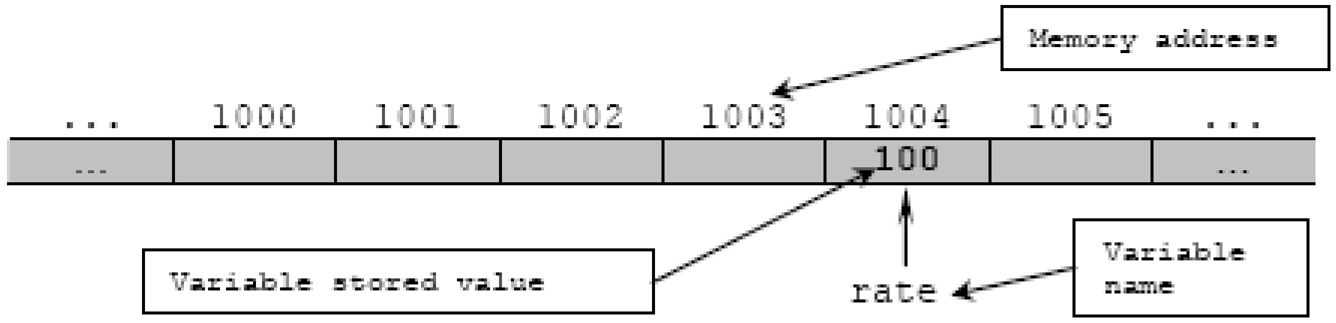
scanf() arguments

- `"%d"` is the input format control string
    - **%** is a special character to mark a conversion format
    - The letter **d** indicates a decimal integer to be input
        - string: %s // strings are char *
        - int: %d
        - char: %c
        - float: %f
- `&integer1` is the variable in which to store the input
    - The ampersand (**&**) is called the address operator in C
    - The name of the variable to be referenced by its address in memory (**integer1**) follows

Computer memory concepts in C

- C allows more direct access to computer memory than many other programming languages
- Variable names refer to data values that will be stored in memory when a program executes
  - my_variable_name /* a data value */
- Variable addresses refer to memory locations that contain data values during program execution
  - &my_variable_name /* a memory address */

Memory location referenced by name

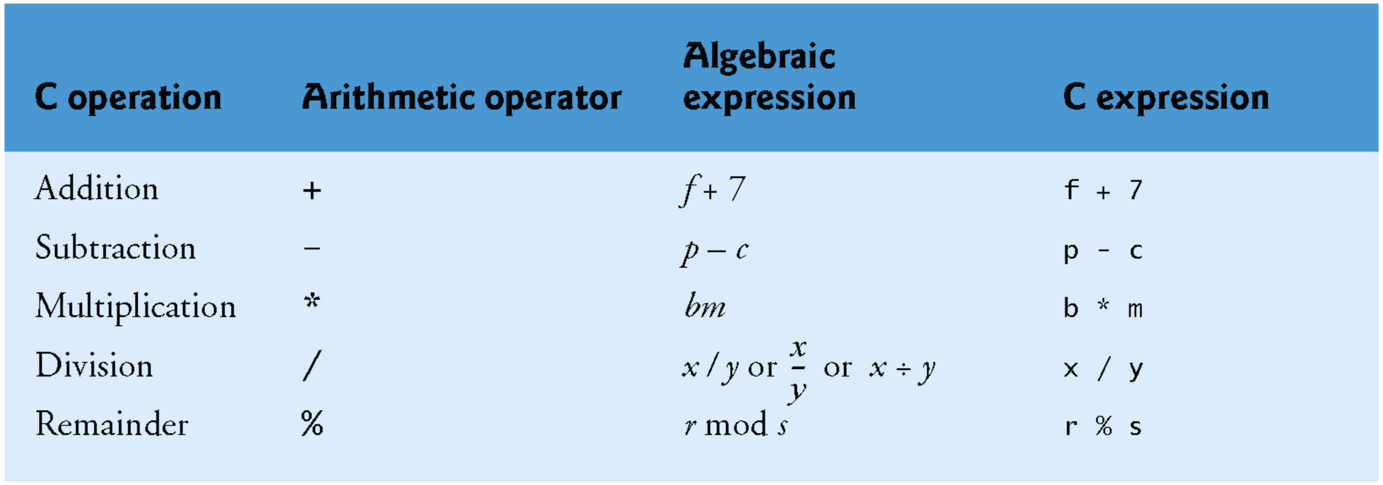


Memory address 1004 contains value 100

Variable name **rate** refers to the value 100

Variable address **&rate** refers to memory location 1004

Arithmetic operators



C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

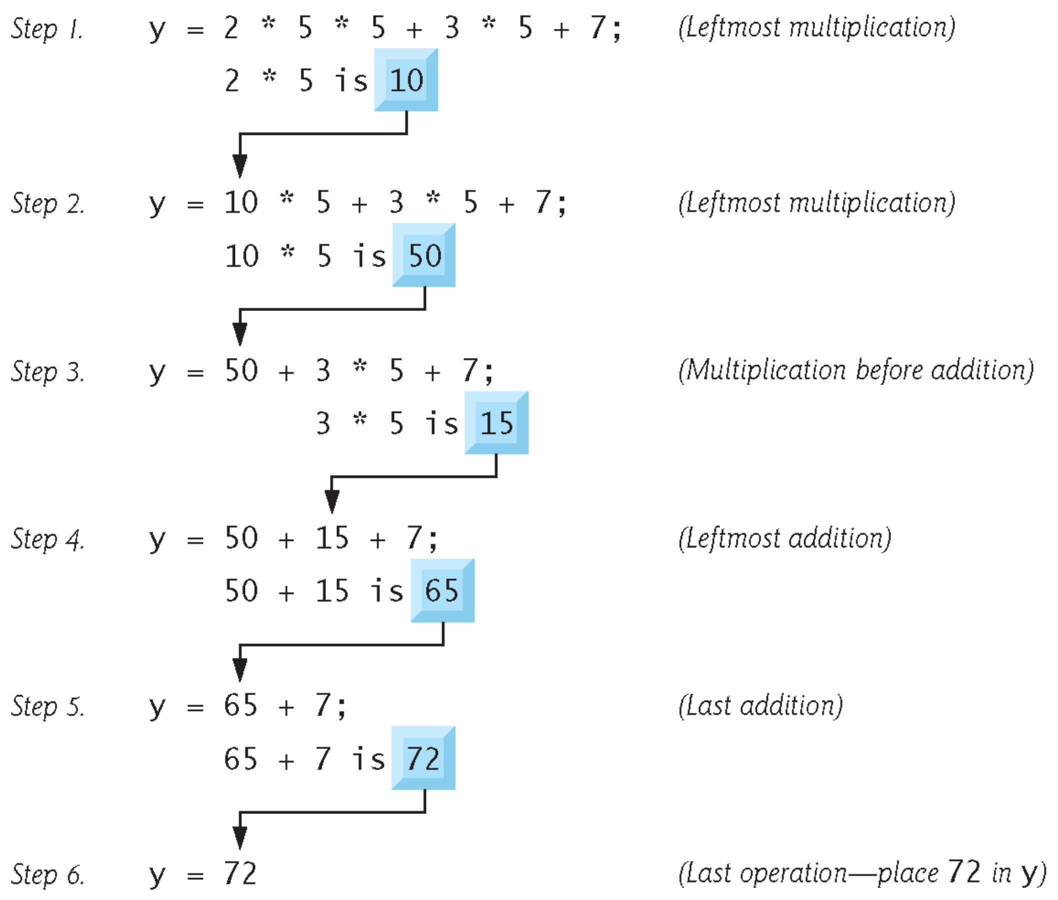
Fig. 2.9 | Arithmetic operators.

Arithmetic operator precedence

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the *innermost* pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
* / %	Multiplication Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated third. If there are several, they're evaluated left to right.
=	Assignment	Evaluated last.

Fig. 2.10 | Precedence of arithmetic operators.

Example of operator precedence



```
Step 1.   y = 2 * 5 * 5 + 3 * 5 + 7;      (Leftmost multiplication)
              2 * 5 is 10

Step 2.   y = 10 * 5 + 3 * 5 + 7;         (Leftmost multiplication)
              10 * 5 is 50

Step 3.   y = 50 + 3 * 5 + 7;             (Multiplication before addition)
                   3 * 5 is 15

Step 4.   y = 50 + 15 + 7;                (Leftmost addition)
              50 + 15 is 65

Step 5.   y = 65 + 7;                     (Last addition)
              65 + 7 is 72

Step 6.   y = 72                          (Last operation—place 72 in y)
```

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

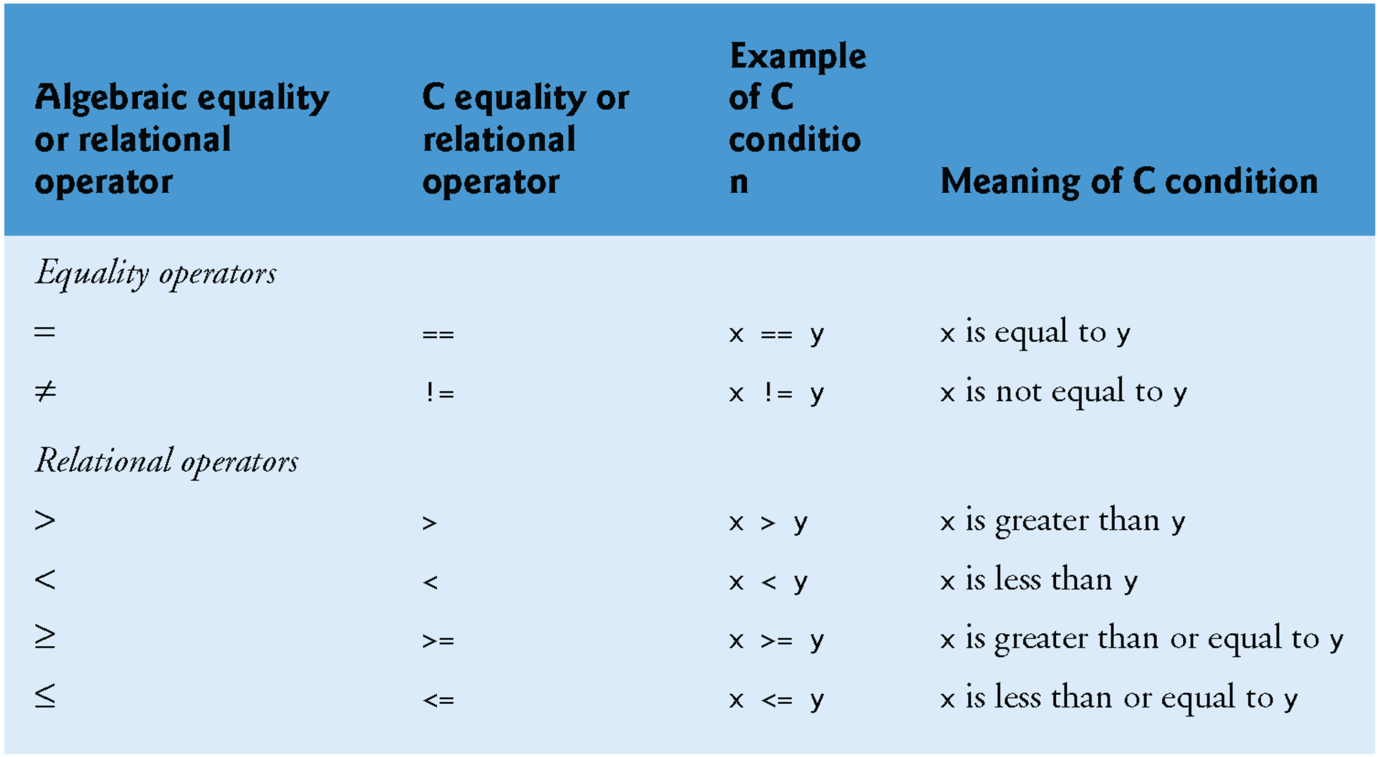
Decision making

- C statements are executed in order from the first function statement to the last
- Decision logic allows conditional execution based on data values to alter the execution flow
- The conditional operators are **if** and **else**
  - if (<condition>) {<body>}
  - if (<condition>) {<body>} else {<alt body>}

Conditional execution flow

- An **if** statement evaluates a logic clause to an integer value of either false (0) or true (>0)
- When the condition is true the body clause of the of the **if** statement executes
- When the condition is false and there is an **else** statement the **else** body clause executes
- Then execution passes to the next statement after the **if/if…else** conditional statements

Relational operators



Algebraic equality or relational operator	C equality or relational operator	Example of C conditio n	Meaning of C condition
Equality operators			
=	`==`	`x == y`	x is equal to y
≠	`!=`	`x != y`	x is not equal to y
Relational operators			
>	`>`	`x > y`	x is greater than y
<	`<`	`x < y`	x is less than y
≥	`>=`	`x >= y`	x is greater than or equal to y
≤	`<=`	`x <= y`	x is less than or equal to y

Fig. 2.12 | Equality and relational operators.

Program example add2ints.c

- This program demonstrates knowledge goals
- scanf() is used to read input values
- printf() is used to write results
- Memory locations using the **&** address operator
- Arithmetic operators used to create results
  - modulo arithmetic using % operator
- Decision making logic used to select output

- [in class exercise]

Knowledge Goals

- Know the fundamental data types of C
- Know how to use variables as arguments to printf() and scanf()
- Understand how data is stored in memory
- Use arithmetic operators to write a formula that assigns its value to a variable
- Understand evaluation of FALSE and TRUE

Next class

- Recommended reading:
  - C_simple_data_types.pdf
  - Chapter 3 from textbook
- Problem Solving
- Reading of interest and studty
  - Appendix C from textbook – Number Systems
  - Program add2ints.c