

An Introduction to the C Programming Language

- Programming and Programming Languages
- The C Programming Language
- A First Program
- Identifiers
- Types
- Constants
- Symbolic Constants
- printf Conversion Specifiers
- Declarations
- Arithmetic Operations
- Relational and Logical Operations
- Bitwise Operators
- Assignment Operators
- Type Conversions and Casts .

Programming and Programming Languages

- ❖ The native language of a computer is binary—ones and zeros—and all instructions and data must be provided to it in this form.
- ❖ Native binary code is called *machine* language.
- ❖ The earliest digital electronic computers were programmed directly in binary, typically via punched cards, plug-boards, or front-panel switches.
- ❖ Later, with the advent of terminals with keyboards and monitors, such programs were written as sequences of hexadecimal numbers, where each hexadecimal digit represents a four binary digit sequence.
- ❖ Developing correct programs in machine language is tedious and complex, and practical only for very small programs.

Programming and Programming Languages

- ❖ In order to express operations more abstractly, *assembly* languages were developed.
- ❖ These languages have simple mnemonic instructions that directly map to a sequence of machine language operations.
- ❖ For example, the MOV instruction moves data into a register, the ADD instruction adds the contents of two registers together.
- ❖ Programs written in assembly language are translated to machine code using an *assembler* program.
- ❖ While assembly languages are a considerable improvement on raw binary, they still very low-level and unsuited to large-scale programming. Furthermore, since each processor provides its own assembler dialect, assembly language programs tend to be non-portable; a program must be rewritten to run on a different machine.
- ❖ The 1950s and 60s saw the introduction of high-level languages

The C Programming Language

- ❖ C is a general-purpose programming language, and is used for writing programs in many different domains, such as operating systems, numerical computing, graphical applications, etc.
- ❖ It is a small language, with just 32 keywords. It provides “high-level” structured programming constructs such as statement grouping, decision making, and looping, as well as “low-level” capabilities such as the ability to manipulate bytes and addresses.
- ❖ *Since C is relatively small, it can be described in a small space, and learned quickly. A programmer can reasonably expect to know and understand and indeed regularly use the entire language*

A First Program

A C program, whatever its size, consists of functions and variables. A function contains statements that specify the computing operations to be done, and variables store values used during the computation

```
/* First C program: Hello World */
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
printf("Hello World!\n");
```

```
}
```

Comments

Start with `/*` and eliminates with `*/`

Header file contains package and library such as `printf` and `scanf`

Entry point to program
May take argument or not

Statement
termination

Start of Entry point block

End of Entry point block

A First Program

- ❖ Comments in C 1 start with `/*` and are terminated with `*/`. They can span multiple lines and are not nestable.
- ❖ Inclusion of a standard library header-file. Most of C's functionality comes from libraries. Headerfiles contain the information necessary to use these libraries, such as function declarations and macros.
- ❖ All C programs have `main()` as the entry-point function. This function comes in two forms:

```
int main(void)  
int main(int argc, char *argv[])
```
- ❖ The braces `{` and `}` delineate the extent of the function block. When a function completes, the program returns to the calling function. In the case of `main()`, the program terminates and control returns to the environment in which the program was executed.
- ❖ The integer return value of `main()` indicates the program's exit status to the environment, with 0 meaning normal termination.
- ❖ This program contains just one statement: a function call to the standard library function `printf()`, which prints a *character string* to standard output Note, `printf()` is not a part of the C language, but a function provided by the standard library (declared in header `stdio.h`)

A First Program

```
1/* Hello World version 2 */  
2#include <stdio.h>  
3  
4int main(void)  
5 {  
6 printf("Hello ");  
7 printf("World!");  
8 printf("\n");  
9 }
```

Operators, Types and Expressions

- ❖ *Variables and constants are the basic data objects manipulated in a program. Declarations list the variables to be used, and state what type they have and perhaps what their initial values are.*
- ❖ *Operators specify what is to be done to them.*
- ❖ *Expressions combine variables and constants to produce new values. The type of an object determines the set of values it can have and what operations can be performed on it*

Reserved Words and Identifiers

- Reserved word
 - Word that has a specific meaning in C
 - Ex: int, return
- Identifier
 - Word used to name and refer to a data element or object manipulated by the program.
 - Identifiers (i.e., variable names, function names, etc) are made up of letters and digits, and are case-sensitive.



RESERVED



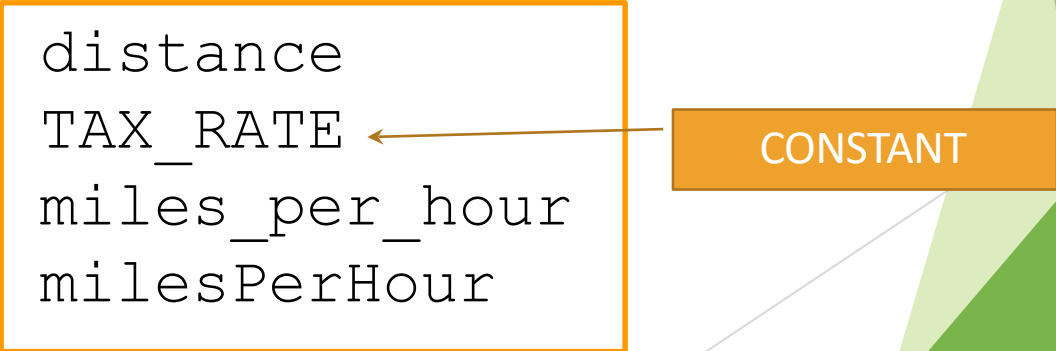
Valid Identifier Names

- Begins with a letter or underscore symbol
- Consists of letters, digits, or underscores only
- Cannot be a C reserved word
- Case sensitive
 - `Total` \neq `total` \neq `TOTAL`
- Examples:

```
distance
milesPerHour
_voltage
goodChoice
high_level
MIN_RATE
```

Identifier Name Conventions

- Standard practice, not required by C language
 - Normally lower case
 - Constants upper case
- Multi-word
 - Underscore between words or
 - Camel case - each word after first is capitalized



A diagram illustrating identifier naming conventions. It features a list of four identifiers: 'distance', 'TAX_RATE', 'miles_per_hour', and 'milesPerHour'. These are enclosed in an orange rectangular box. To the right of this box is an orange rectangular label with the word 'CONSTANT' in white capital letters. A thin orange arrow points from the 'CONSTANT' label to the 'TAX_RATE' identifier, indicating that it is a constant.

```
distance  
TAX_RATE  
miles_per_hour  
milesPerHour
```

CONSTANT

Variable



- Name is a valid identifier name
- Is a memory location where a value can be stored for use by a program
- Value can change during program execution
- Can hold only one value
 - Whenever a new value is placed into a variable, the new value replaces the previous value.

Variables Names



- C: Must be a valid identifier name
- C: Variables must be declared with a name and a data type *before* they can be used in a program
- Should not be the name of a standard function or variable
- Should be descriptive; the name should be reflective of the variable's use in the program
 - For class, make that must be descriptive except subscripts
- Abbreviations should be commonly understood
 - Ex. `amt` = amount

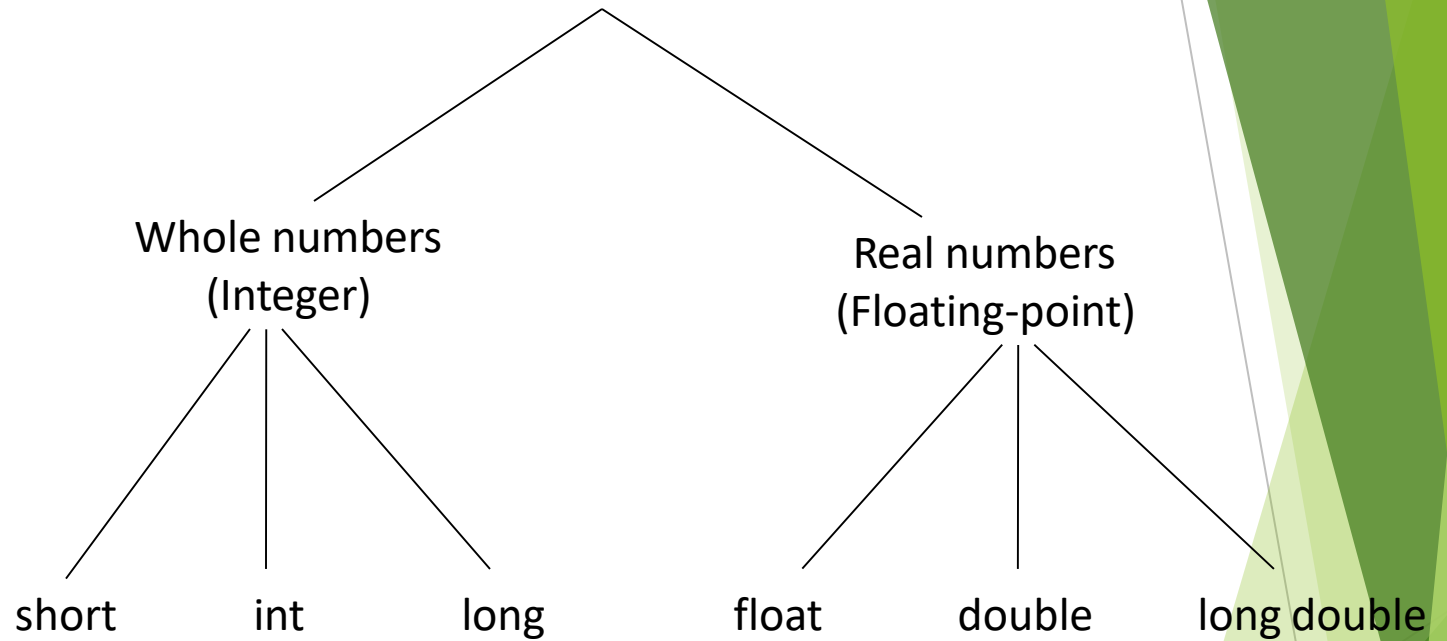
Variable/Named Constant Declaration Syntax

`optional_modifier data_type name_list;`

- *optional_modifier* – type modifier
 - Used to distinguish between **signed** and **unsigned** integers
 - The default is signed
 - Used to specify size (**short**, **long**)
 - Used to specify named constant with **const** keyword
- *data_type* - specifies the type of value; allows the compiler to know what operations are valid and how to represent a particular value in memory
- *name_list* – program identifier names
- Examples:

```
int test-score;  
const float TAX_RATE = 6.5;
```

Numeric Data Types



Data Types and Typical Sizes

Type Name	Memory Used	Size Range	Guarantee
short (= short int)	2 bytes	-32,768 to 32,767	16 bits
int	4 bytes	-2,147,483,648 to 2,147,483,647	32 bits
long (= long int)	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	64bits
float	4 bytes	approximately 10^{-38} to 10^{38}	6 digits
double	8 bytes	approximately 10^{-308} to 10^{308}	10 digits
long double	10 bytes	approximately 10^{-4932} to 10^{4932}	10 digits
char	8-bits (1 byte)		8BITS

Determining Data Type Size

- sizeof operator
 - Returns size of operand in bytes
 - Operand can be a data type
- Examples:

```
sizeof(int)  
sizeof(double)
```



Data Types and Typical Sizes

```
#include <stdio.h>
#include <limits.h> /* integer specifications */
#include <float.h> /* floating-point specifications */

/* Look at range limits of certain types */
int main (void)
{
    printf("Integer range:\t%d\t%d\n", INT_MIN, INT_MAX);
    printf("Long range:\t%ld\t%ld\n", LONG_MIN, LONG_MAX);
    printf("Float range:\t%e\t%e\n", FLT_MIN, FLT_MAX);
    printf("Double range:\t%e\t%e\n", DBL_MIN, DBL_MAX);
    printf("Long double range:\t%e\t%e\n", LDBL_MIN, LDBL_MAX);
    printf("Float-Double epsilon:\t%e\t%e\n", FLT_EPSILON, DBL_EPSILON);
}
```

Characters

Type Name	Memory Used	Sample Size Range
char	1 byte	All ASCII characters

ASCII = American Standard Code for Information Interchange

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

www.asciitable.com

Boolean Data Type

- Data type: `_Bool`
 - Can only store 0 & 1
 - Non zero value will be stored as 1
- Data type : `bool`
 - `<stdbool.h>` defines `bool`, `true`, and `false`
- Any expression
 - 0 is false
 - Non-zero is true



Variable Declaration Examples

```
int age;

short first_reading;
short int last_reading;

long first_ssn;
long int last_ssn;

float interest_rate;
double division_sales;

char grade, midInitial;
```

Assigning Values to Variables

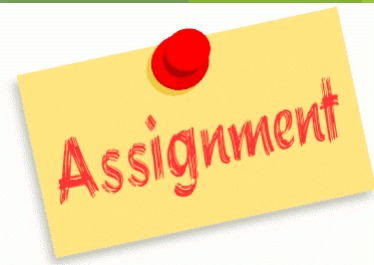
- Allocated variables without initialization have an undefined value.
- We will use three methods for assigning a value to a variable
 - Initial value
 - In the declaration statement
 - Processing
 - the assignment statement
 - Input
 - scanf function

Initializing Variables

- Initializing variables in declaration statements

```
int age = 22;  
double rate = 0.75;  
char vowel = 'a';  
int count = 0, total = 0;
```

Assignment Operator =



- Assigns a value to a variable
- Binary operator (has two operands)
- Not the same as "equal to" in mathematics
- General Form:

`l_value = r_value`

- Most common examples of l_values (left-side)
 - A simple variable
 - A pointer dereference (in later chapters)
- r_values (right side) can be any valid expression
- Assignment expression has value of assignment
 - Allows us to do something like
`a = b = 0;`

Example Assignment Statement

- Statement



```
x = y + 5;
```

5 is literal value
or constant

- Means:

Evaluate the expression on the right and put the result in the memory location named x

- If the value stored in y is 18,
then 23 will be stored in x

Other Example Assignments

- Example:

```
distance = rate * time;
```

l_value: distance

r_value: rate * time

- Other Examples:

```
pay = 65.75;  
hourly_rate = pay / hours;
```



Terminal Output

What can be output?

- Any data can be output to standard output (stdout), the terminal display screen
 - Literal values
 - Variables
 - Constants
 - Expressions (which can include all of above)
- printf function:
The values of the variables are passed to printf

Syntax: printf function

```
printf(format_string, expression_list)
```

- Format_string specifies how expressions are to be printed
 - Contains placeholders for each expression
 - Placeholders begin with % and end with type
- Expression list is a list of zero or more expressions separated by commas
- Returns number of characters printed

Typical Integer Placeholders

- %d or %i - for integers, %l for long

```
printf("%d", age);  
printf("%l", big_num);
```

- %o - for integers in octal

```
printf("%o", a);
```

- %x – for integers in hexadecimal

```
printf("%x", b);
```

Floating-point Placeholders

- %f, %e, %g – for float
 - %f – displays value in a standard manner.
 - %e – displays value in scientific notation.
 - %g – causes printf to choose between %f and %e and to automatically remove trailing zeroes.
- %lf – for double (the letter l, not the number 1)

Printing the value of a variable

- We can also include literal values that will appear in the output.
 - Use two %'s to print a single percent

\n is new line

```
printf("x = %d\n", x);  
printf("%d + %d = %d\n", x, y, x+y);  
printf("Rate is %d%%\n", rate*100);
```

Output Formatting Placeholder

`%[flags][width][.precision][length]type`

- Flags
 - left-justify
 - + generate a plus sign for positive values
 - # puts a leading 0 on an octal value and 0x on a hex value
 - 0 pad a number with leading zeros
- Width
 - Minimum number of characters to generate
- Precision
 - Float: Round to specified decimal places

Output Formatting Placeholder

`%[flags][width][.precision][length]type`

- Length
 - l long
- Type
 - d, i decimal unsigned int
 - f float
 - x hexadecimal
 - o octal
 - % print a %

Output Formatting Placeholder

`%[flags][width][.precision][length]type`

- Examples:

```
printf("[%5d] [%+05d] [%#5o] [%#7x]\n",  
       123, 123, 123, 123);  
printf("[%f] [%5.2f] [%5.0f%%]\n",  
       123.456, 123.456, 123.456);
```

```
[ 123] [+0123] [ 0173] [ 0x7b]  
[123.456000] [123.46] [ 123%]
```

Return from printf

- A successful completion of printf returns the number of characters printed. Consequently, for the following:

```
int num1 = 55;  
int num2 = 30;  
int sum = num1 + num2;  
int printCount;  
printCount = printf("%d + %d = %d\n", num1, num2, sum);
```

if printf() is successful,
the value in printCount should be 13.

Literals / Literal Constants

- Literal – a name for a specific value
- Literals are often called constants
- Literals do not change value

Literal

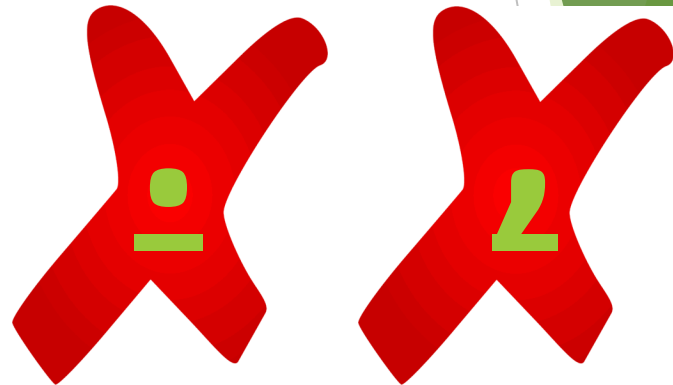
Integer Constants

- Must not contain a decimal point
- Must not contain a comma
- Examples

-25

68

17895



Integer Constants

- May be expressed in several ways

decimal number 120

hexadecimal number 0x78

octal number 0170

ASCII encoded character 'x'

- All of the above represent the 8-bit byte whose value is 01111000

119	77	167	w	W
120	78	170	x	X
121	79	171	y	Y

Integer Constants

- Constants of different representations may be intermixed in expressions:
 - Examples

```
x = 5 + 'a' - 011 + '\n';  
x = 0x51 + 0xc + 0x3d + 0x8;
```

Floating Point Constants

- Contain a decimal point.
- Must not contain a comma
- Can be expressed in two ways

decimal number: **23.8** **4.0**

scientific notation: **1.25E10**



char Constants

- Enclosed in apostrophes, single quotes

- Examples:

'a'

'A'

'\$'

'2'

'+'

- Format specification: %c

String Constants

- Enclosed in quotes, double quotes

- Examples:

`"Hello"`

`"The rain in Spain"`

`"x"`

- Format specification/placeholder: %s

Terminal Input

- We can put data into variables from the standard input device (stdin), the terminal keyboard
- When the computer gets data from the terminal, the user is said to be acting interactively.
- Putting data into variables from the standard input device is accomplished via the use of the scanf function



Keyboard Input using scanf

- General format

`scanf(format-string, address-list)`

- Example

```
scanf("%d", &age);
```

& (address of operator)
is required

& € \$ ¢

- The format string contains placeholders (one per address) to be used in converting the input.
 - %d – Tells *scanf* that the program is expecting an ASCII encoded integer number to be typed in, and that *scanf* should convert the string of ASCII characters to internal binary integer representation.
- Address-list: List of memory addresses to hold the input values

Addresses in scanf()

```
scanf("%d", &age);
```

- Address-list must consist of addresses only
 - scanf() puts the value read into the memory address
 - The variable, age, is not an address; it refers to the *content* of the memory that was assigned to age
- & (address of) operator causes the **address of the variable** to be passed to *scanf* rather than the value in the variable
- Format string should consist of a placeholder for each address in the address-list

Return from scanf()

- A successful completion of scanf() returns the number of input values read. Returns EOF if hits end-of-file reading one item.

Consequently, we could have

```
int dataCount;  
dataCount = scanf("%d %d", &height, &weight);
```

- If scanf() is successful, the value in dataCount should be 2
- Spaces or new lines separate one value from another

Keyboard Input using scanf

- When using scanf for the terminal, it is best to first issue a prompt

```
printf("Enter the person's age: ");  
scanf("%d", &age);
```

- Waits for user input, then stores the input value in the memory space that was assigned to number.
- Note: '\n' was omitted in printf
 - Prompt 'waits' on same line for keyboard input.
- Including printf prompt before scanf maximizes user-friendly input/output

scanf Example

```
int main() {  
    // declare variables  
    int x;  
    int y;  
    int sum;  
  
    // read values for x and y from standard input  
    printf("Enter value for x: ");  
    scanf("%d", &x);  
  
    printf("Enter value for y: ");  
    scanf("%d", &y);  
  
    sum = x + y;  
  
    // print  
    printf("x = %d\n", x);  
    printf("y = %d\n", y);  
    printf("x + y = %d\n", sum);  
  
    printf("%d + %d = %d\n", x, y, sum);  
    printf("%d - %d = %d\n", x, y, (x - y));  
    printf("%d * %d = %d\n", x, y, (x * y));  
    return 0;  
}
```


Input using scanf()

- Instead of using scanf() twice, we can use one scanf() to read both values.

```
int main() {  
    // declare variables  
    int x;  
    int y;  
    int sum;  
  
    // read values for x and y from standard input  
    printf("\n");  
    printf("Enter values for x and y: ");  
    scanf("%d %d", &x, &y);  
  
    sum = x + y;  
  
    // print  
    printf("x = %d\n", x);  
    printf("y = %d\n", y);  
    printf("x + y = %d\n", sum);  
  
    printf("%d + %d = %d\n", x, y, sum);  
    printf("%d - %d = %d\n", x, y, (x - y));  
    printf("%d * %d = %d\n", x, y, (x * y));  
  
    printf("\n");  
    return 0;  
}
```

Format Placeholder for Input

- When reading data, use the following format specifiers / placeholders
 - %d - for integers, no octal or hexadecimal
 - %i – for integers allowing octal and hexadecimal
 - %f - for float
 - %lf – for double (the letter l, not the number 1)
- Do not specify width and other special printf options

Executable Code

- Expressions consist of legal combinations of
 - constants
 - variables
 - operators
 - function calls

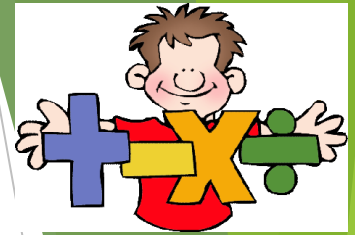


Executable Code

- Operators

- Arithmetic: `+, -, *, /, %`
- Relational: `==, !=, <, <=, >, >=`
- Logical: `!, &&, ||`
- Bitwise: `&, |, ~, ^`
- Shift: `<<, >>`

Arithmetic



- Rules of operator precedence (arithmetic ops):

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 are evaluated left to right.
*, /, or %	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

- Average $(a + b + c) / 3$?

Precedence Example

- Find the average of three variables a, b and c

Do not use: $a + b + c / 3$

Use: $(a + b + c) / 3$

The Division Operator

- Generates a result that is the same data type of the largest operand used in the operation.
- Dividing two integers yields an integer result. Fractional part is truncated.

$$5 / 2 \rightarrow 2$$

$$17 / 5 \rightarrow 3$$

➤ Watch out: You will not be warned!

$$\begin{array}{r} 193 \\ 5 \overline{)965} \\ \underline{-5} \\ 46 \\ \underline{-45} \\ 15 \\ \underline{-15} \\ 0 \end{array}$$

$15 \div 5 = 3$

The Division Operator

- Dividing one or more decimal floating-point values yields a decimal result.

5.0 / 2 → 2.5

4.0 / 2.0 → 2.0

17.0 / 5.0 → 3.4

The modulus operator: %

- % modulus operator returns the remainder

$$7 \% 5 \rightarrow 2$$

$$5 \% 7 \rightarrow 5$$

$$12 \% 3 \rightarrow 0$$

A handwritten long division of 965 by 5. The divisor 5 is on the left, and the dividend 965 is on the right. The quotient 193 is written above the dividend. The steps are: 5 goes into 9 one time (1), 5 goes into 6 one time (1), and 5 goes into 5 one time (3). The remainders are 4, 1, and 0 respectively. A red arrow points from the final remainder 15 to the text $15 \div 5 = 3$.

$$\begin{array}{r} 193 \\ 5 \overline{) 965} \\ \underline{-5} \\ 46 \\ \underline{-45} \\ 15 \end{array} \quad 15 \div 5 = 3$$

Evaluating Arithmetic Expressions

- Calculations are done 'one-by-one' using precedence, left to right within same precedence
 - $11 / 2 / 2.0 / 2$ performs 3 separate divisions.
 1. $11 / 2 \rightarrow 5$
 2. $5 / 2.0 \rightarrow 2.5$
 3. $2.5 / 2 \rightarrow 1.25$



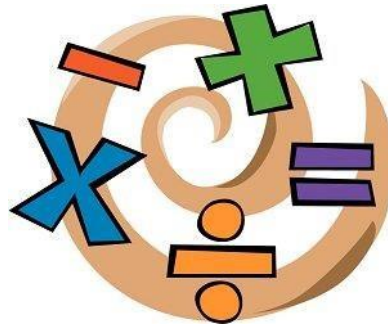
Arithmetic Expressions

math expression

$$\frac{a}{b}$$

$$2x$$

$$\frac{x-7}{2+3y}$$



C expression

$$a/b$$

$$2*x$$

$$(x-7)/((2+3)*y)$$

Evaluating Arithmetic Expressions

$2 * (-3)$	-6
$4 * 5 - 15$	5
$4 + 2 * 5$	14
$7 / 2$	3
$7 / 2.0$	3.5
$2 / 5$	0
$2.0 / 5.0$	0.4
$2 / 5 * 5$	0
$2.0 + 1.0 + 5 / 2$	5.0
$5 \% 2$	1
$4 * 5 / 2 + 5 \% 2$	11

Data Assignment Rules

- In C, when a floating-point value is assigned to an integer variable, the decimal portion is truncated.

```
int grams;  
grams = 2.99;    // 2 is assigned to variable grams!
```

- Only integer part 'fits', so that's all that goes
- Called 'implicit' or 'automatic type conversion'



Arithmetic Precision

- Precision of Calculations
 - VERY important consideration!
 - Expressions in C might not evaluate as you 'expect'!
 - 'Highest-order operand' determines type of arithmetic 'precision' performed
 - Common pitfall!
 - Must examine each operation

Type Casting

change

- Casting for Variables
 - Can add '.0' to literals to force precision arithmetic, but what about variables?
 - We can't use '`myInt.0`'!
- type cast – a way of changing a value of one type to a value of another type.
- Consider the expression $1/2$: In C this expression evaluates to 0 because both operands are of type integer.

Type Casting

1 / 2.0 gives a result of 0.5

Given the following:

```
int m = 1;  
int n = 2;  
int result = m / n;
```

result is 0, because of integer division

Type Casting

- To get floating point-division, you must do a type cast from int to double (or another floating-point type), such as the following:

```
int m = 1;  
int n = 2;  
double doubleAnswer = (double) m / n;
```



Type cast operator

- This is different from `(double) (m/n)`

Type Casting

- Two types of casting

- Implicit – also called ‘Automatic’

- Done for you, automatically

`17 / 5.5`

This expression causes an ‘implicit type cast’ to take place, casting the 17 → 17.0

- Explicit type conversion

- Programmer specifies conversion with cast operator

`(double)17 / 5.5`

`(double) myInt / myDouble`

Abbreviated/Shortcut Assignment Operators

- Assignment expression abbreviations

`a = a + 3;` can be abbreviated as `a += 3;`
using the addition assignment operator

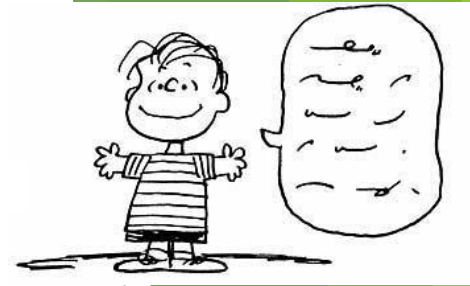
- Examples of other assignment operators include:

Assignment	Shortcut
<code>d = d - 4</code>	<code>d -= 4</code>
<code>e = e * 5</code>	<code>e *= 5</code>
<code>f = f / 3</code>	<code>f /= 3</code>
<code>g = g % 9</code>	<code>g %= 9</code>

- Shortcut



Shorthand Operators



- Increment & Decrement Operators
 - Just short-hand notation
 - Increment operator, ++
`intVar++;` is equivalent to
`intVar = intVar + 1;`
 - Decrement operator, --
`intVar--;` is equivalent to
`intVar = intVar - 1;`

Shorthand Operators: Two Options

- Post-Increment

x++

- Uses current value of variable,
THEN increments it

- Pre-Increment

++x

- Increments variable first,
THEN uses new value

POST

PRE

Shorthand Operators: Two Options

- 'Use' is defined as whatever 'context' variable is currently in
- No difference if 'alone' in statement:
`x++;` and `++x;` → identical result

Post-Increment in Action

POST

- Post-Increment in Expressions:

```
int n = 2;  
int result;  
result = 2 * (n++);  
printf("%d\n", result);  
printf("%d\n", n);
```

- This code segment produces the output:
4
3
- Since post-increment was used

Pre-Increment in Action

- Now using pre-increment:

```
int n = 2;  
int result;  
result = 2 * (++n);  
printf("%d\n", result);  
printf("%d\n", n);
```

- This code segment produces the output:
6
3
- Because pre-increment was used

PRE

Quiz

- Write a C program to convert a character to integer.

Quiz

- Write a C program to return the quotient and remainder of a division.

Quiz

1. C Program to Swap Two Numbers