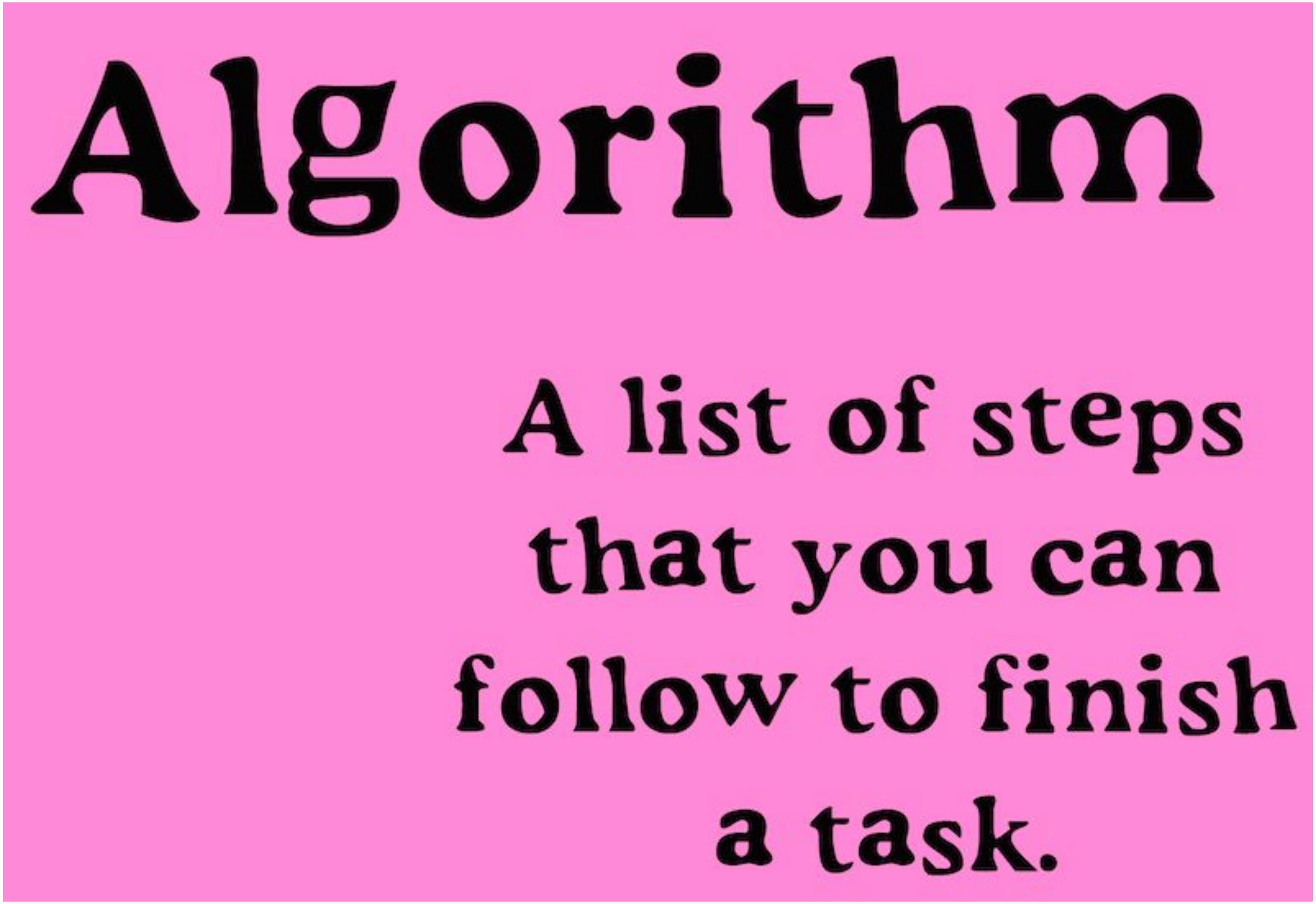
Fundamentals of Programming

Lecture 2

Akara Supratak
Jidapa Kraisangka
Pilailuck Panphattarasap

Recap



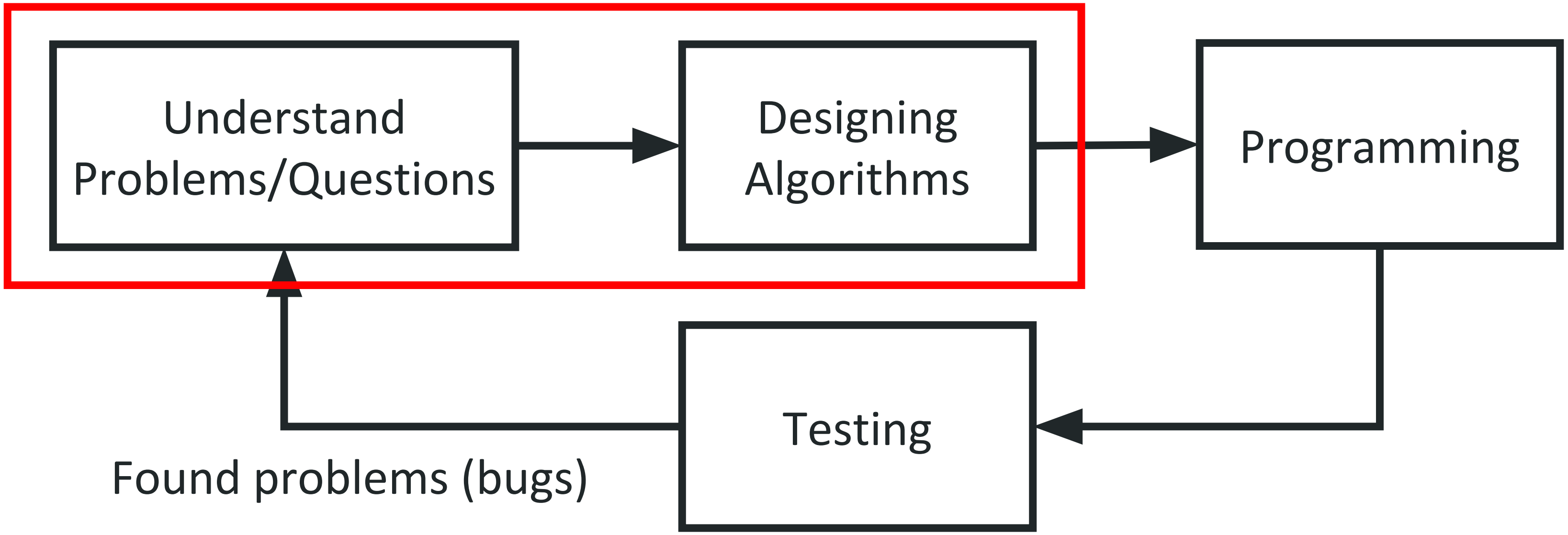
Algorithm

A list of steps that you can follow to finish a task.

Solve a Problem with Algorithm

Algorithm: procedure for solving a problem in terms of

1. The actions to be executed
2. The order



Compile and Run a C Program

1. Writes a C program in VS Code
2. Use a compiler (e.g., gcc) to translate the code into a machine language program (output file).

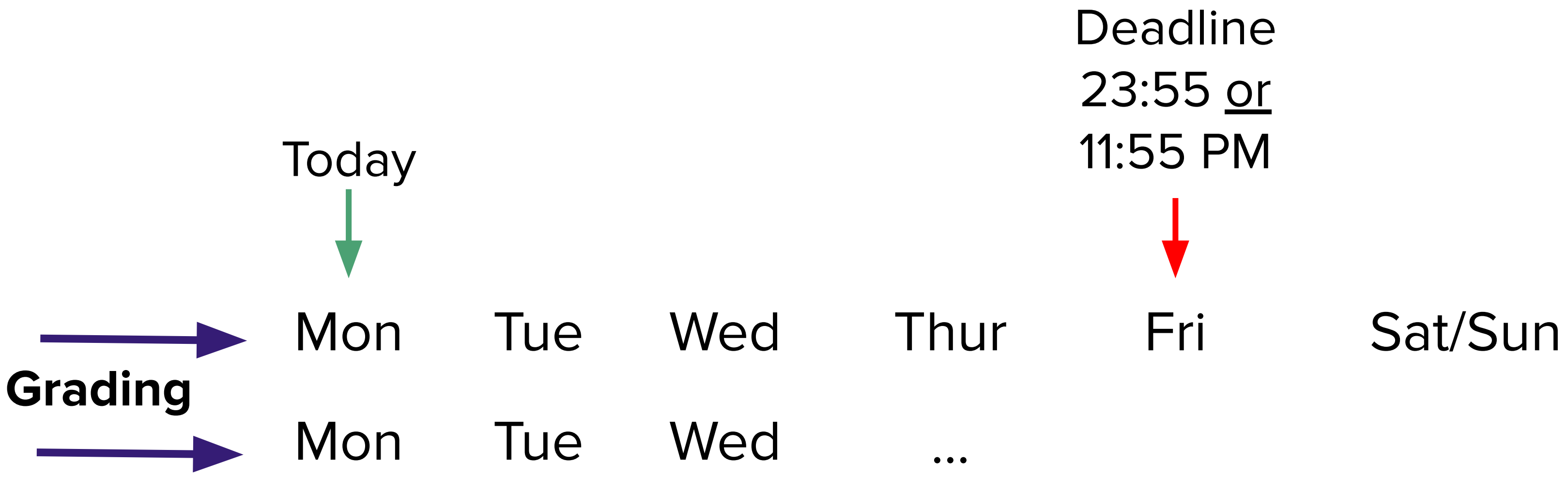
```
gcc <filename.c> -o <filename>
```

3. Run the output file

```
./<filename>
```

********** Lab Assignments **********

- Every week you have approx. 2 hour to do the lab assignment in class and can submit until Friday
  - Submit your code to **PC^2** until getting “YES”
  - Call me or LAs for Q&A your code



Deadline 23:55 or 11:55 PM

Today

Mon Tue Wed Thur Fri Sat/Sun

Grading

Mon Tue Wed …

Make-up Class Confirmation

NO CLASS on

- Mon Sep 30: MU Grand (Graduation) Rehearsal
- Mon Oct 13 King Bhumibol Memorial Day

MAKE-UP CLASS on

- Wed Oct 9 Afternoon *(Lecture 8)*
- Wed Dec 4 Afternoon *(Lecture 15)*

Project

- Will be done a in Group of 4 - 5 people, i.e., 9 groups of five and 6 groups of four
- In MyCourse: Project Section, **fill in** your group members according to YOUR CLASS SECTION sheet.
- **Deadline to fill in**: Monday Oct 2, 2019 12:05 PM (noon)

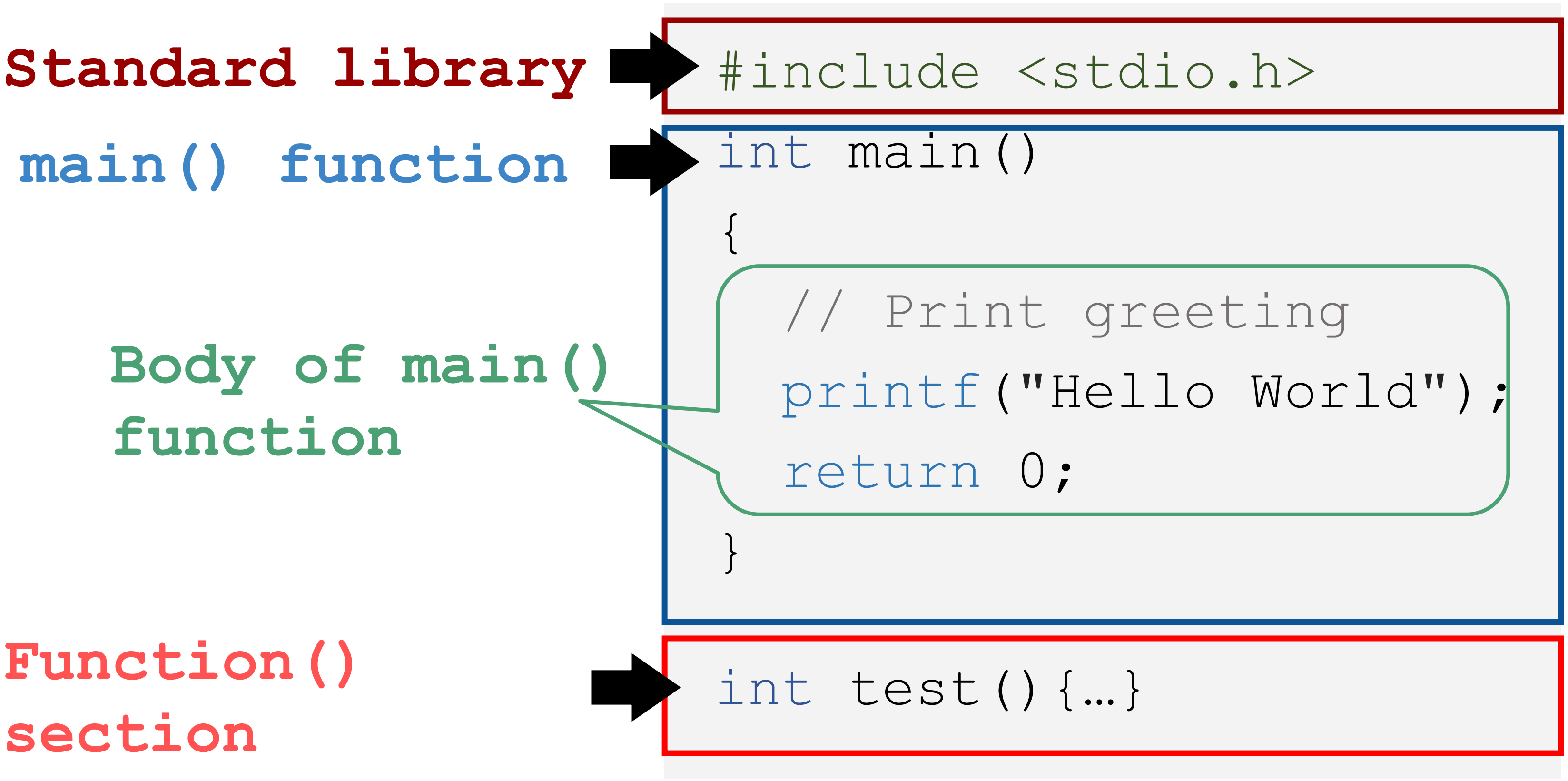
  You cannot modify the member sheet after deadline
- The remaining students will be randomly assigned to groups :)
- We will discuss the description later in class

Today Topic

- Basic component in C programs
- Variables
- Data types
- Arithmetic, Relational and Logical Operations
- Error and Debugging

Basic C Programming

Structure of a Basic C program



Standard library

main() function

Body of main() function

Function() section

```
#include <stdio.h>
int main()
{
  // Print greeting
  printf("Hello World");
  return 0;
}
int test(){…}
```

Structure of a Basic C program

1 **Standard library** →

```
#include <stdio.h>
int main()
{
   // Print greeting
   printf("Hello World");
   return 0;
}

int test(){…}
```

1 Standard Library

- Instruction: a commands for a single operation.
- Function: a named section of program that contain a set of instructions used to perform a specific task.
- C provides a comprehensive set of functions, stored in a set of files known as the standard library e.g., stdio.h, math.h, …

Standard Library (cont.)

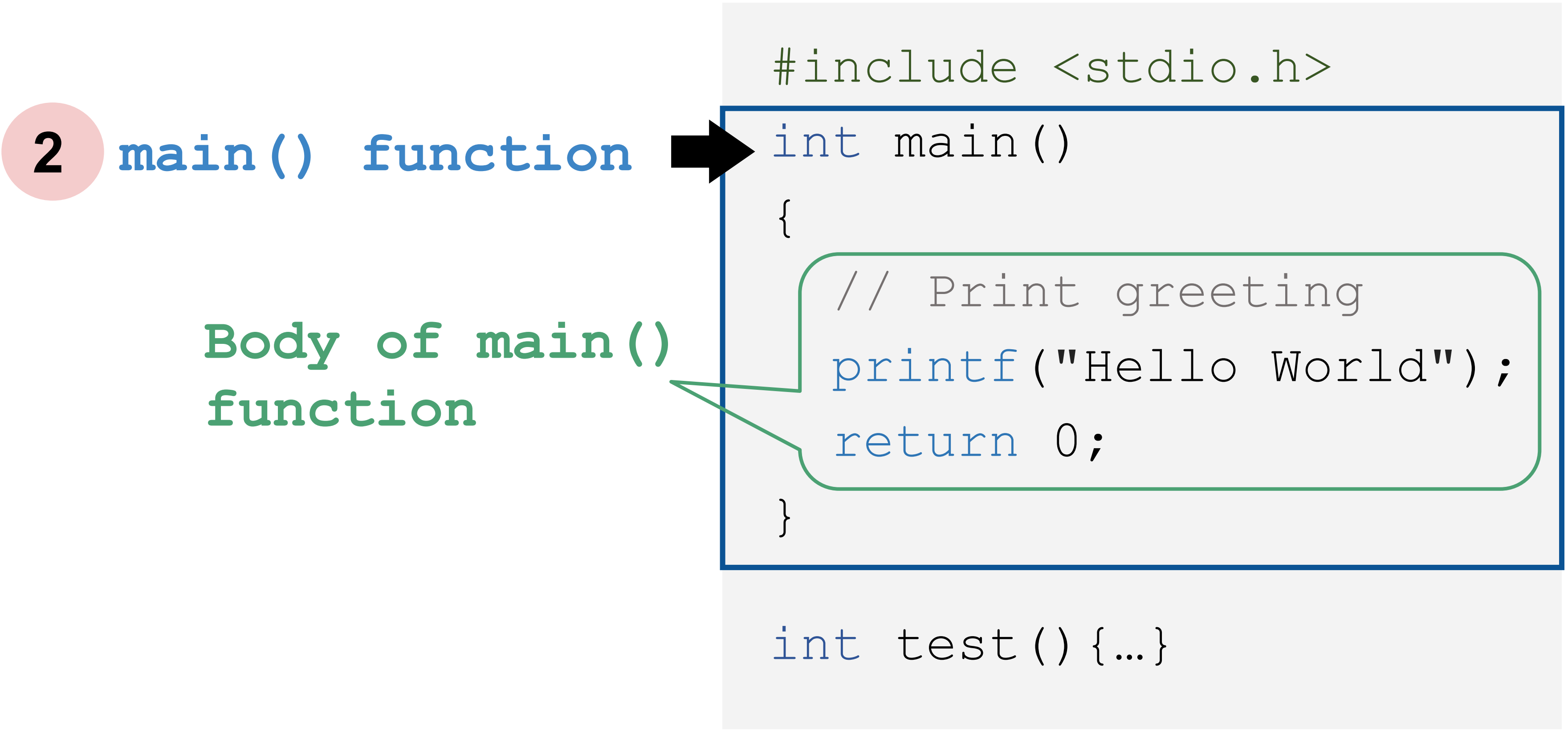
`#include <lib.h>`

A library that defines a set of “**functions**” that you can use.

For example, `printf()` and `scanf()` are defined in `stdio.h`

```
#include <stdio.h>
int main()
{
  // Print greeting
  printf("Hello World");
  return 0;
}

int test(){…}
```

Structure of a Basic C program



2 main() function

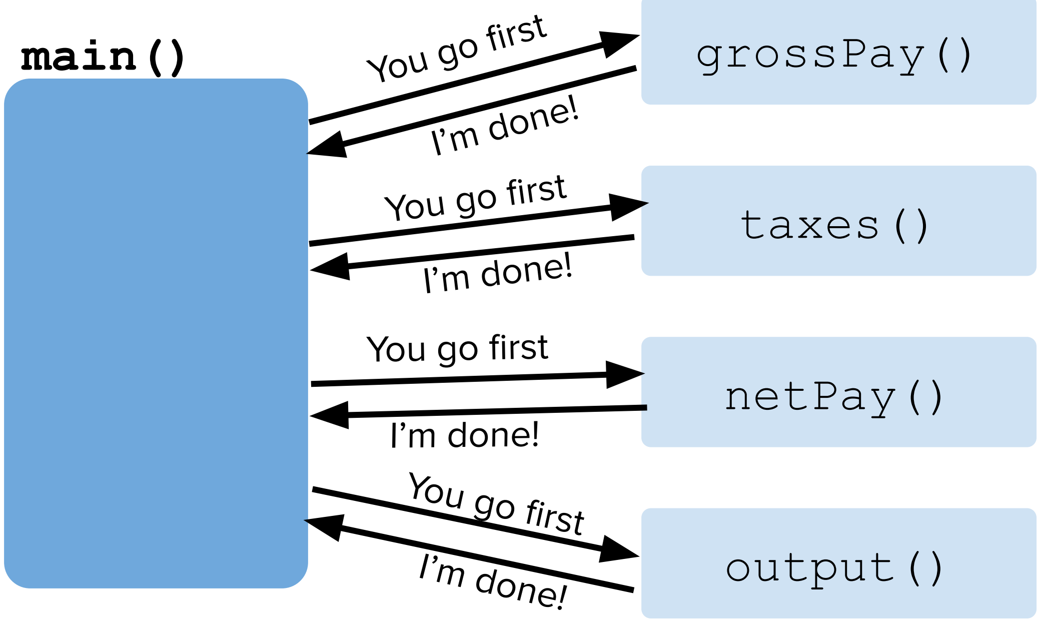
Body of main() function

```
#include <stdio.h>
int main()
{
  // Print greeting
  printf("Hello World");
  return 0;
}

int test(){…}
```

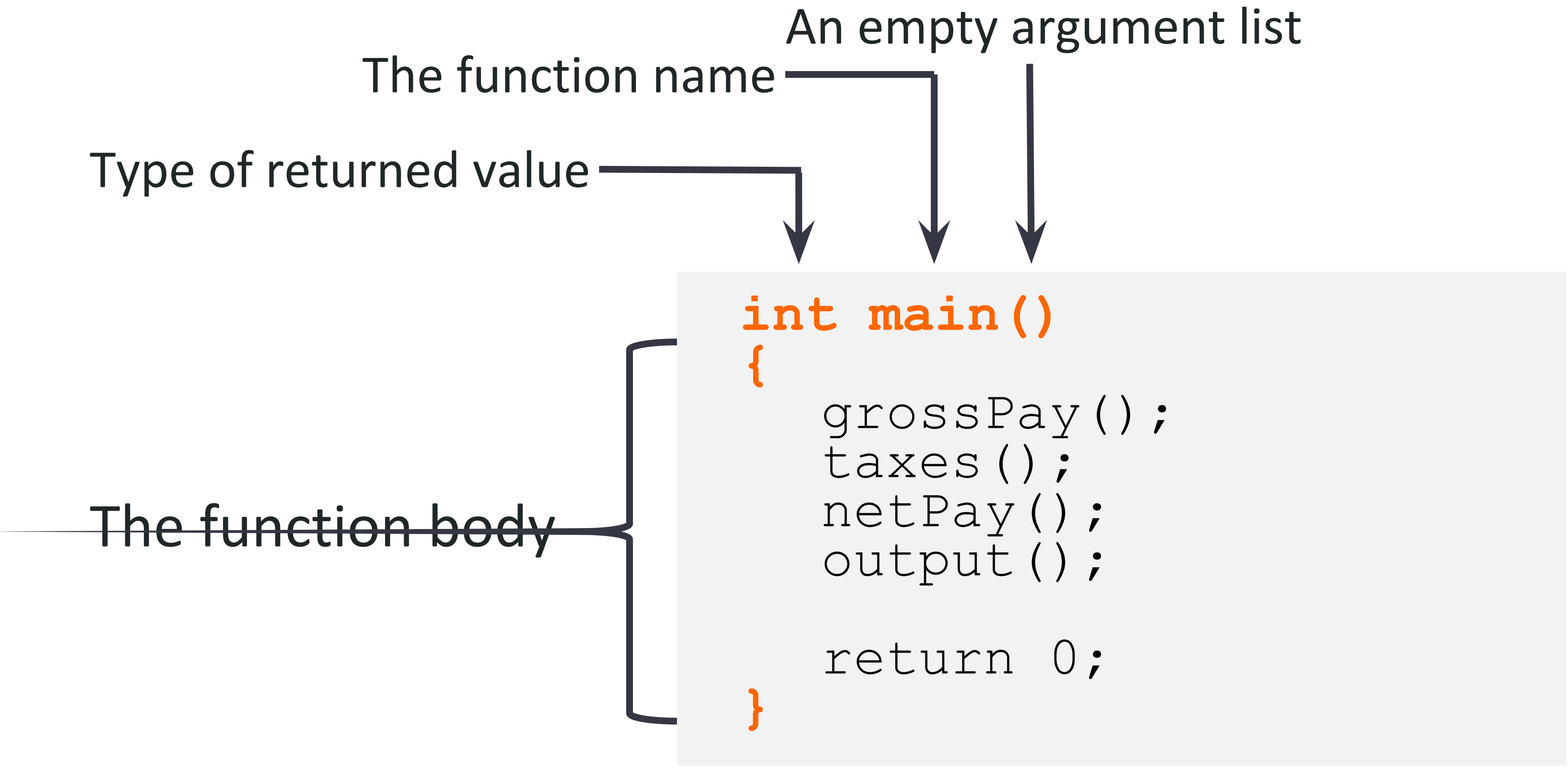
2 `main()` function

- A mandatory function in C, where a program **start** to execute.
- Have all features of function.

Sometimes `main()` referred to as a "**driver function**"



main() function (cont.)



An empty argument list

The function name

Type of returned value

The function body

```
int main()
{
    grossPay();
    taxes();
    netPay();
    output();

    return 0;
}
```

`main()` function (cont.)

`main()`

`int main(){…}` is a "box of statements" that a program will start executing until the end or reaching **`return …;`** statement.

In the example, it starts with `printf()`.

```
#include <stdio.h>
int main()
{
   // Print greeting
   printf("Hello World");
   return 0;
}

int test(){…}
```

Statement

statement;

The algorithm of a program. You should be able to convert pseudocode or flowchart into a list of statements.

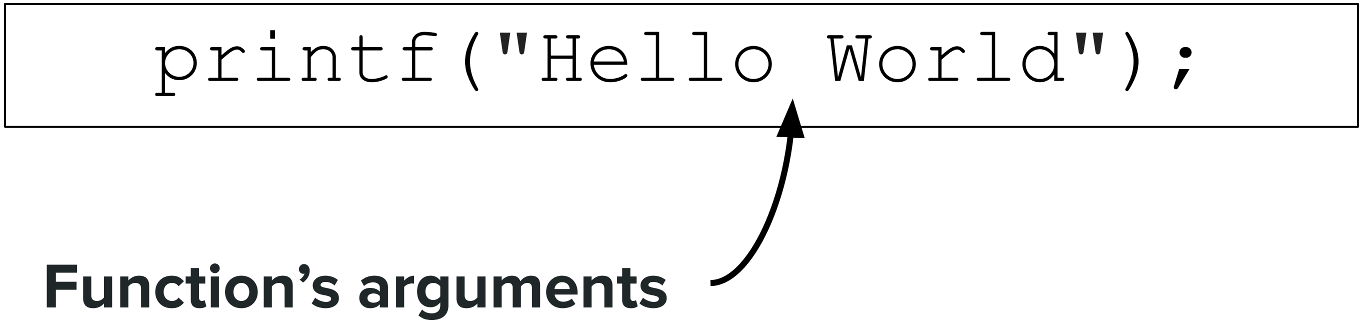
Statement can be:

- Single instruction
- A set of instruction

```
#include <stdio.h>
int main()
{
   // Print greeting
   printf("Hello World");
   return 0;
}

int test(){…}
```

`printf()` function

- A function that formats data and sends it to the standard system display device (monitor)
- Inputting data or messages to a function is called “passing data to the function”

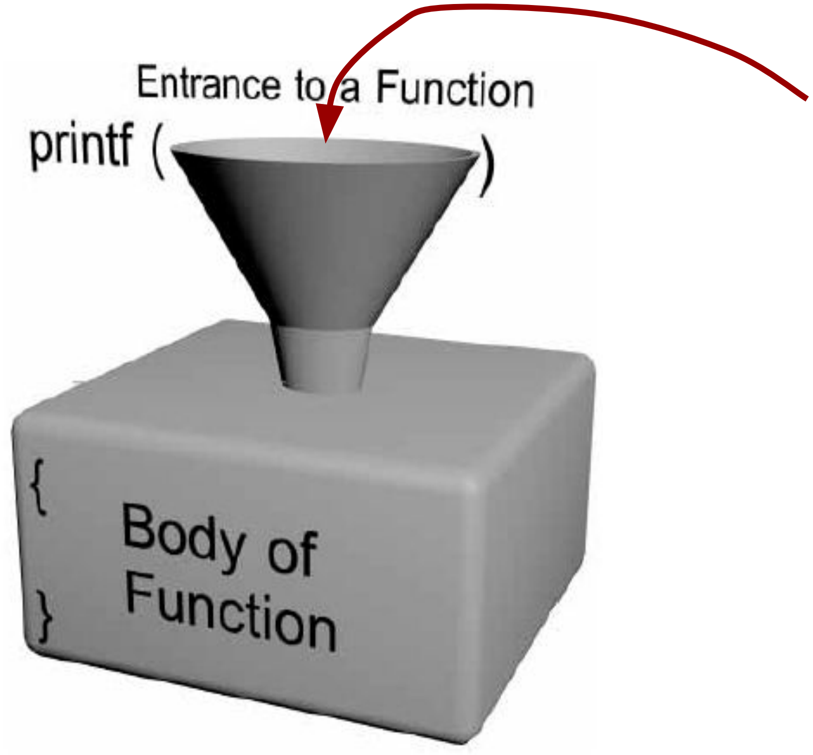


```
printf("Hello World");
```

Function’s arguments

`printf()` function

```
printf("Hello World");
```



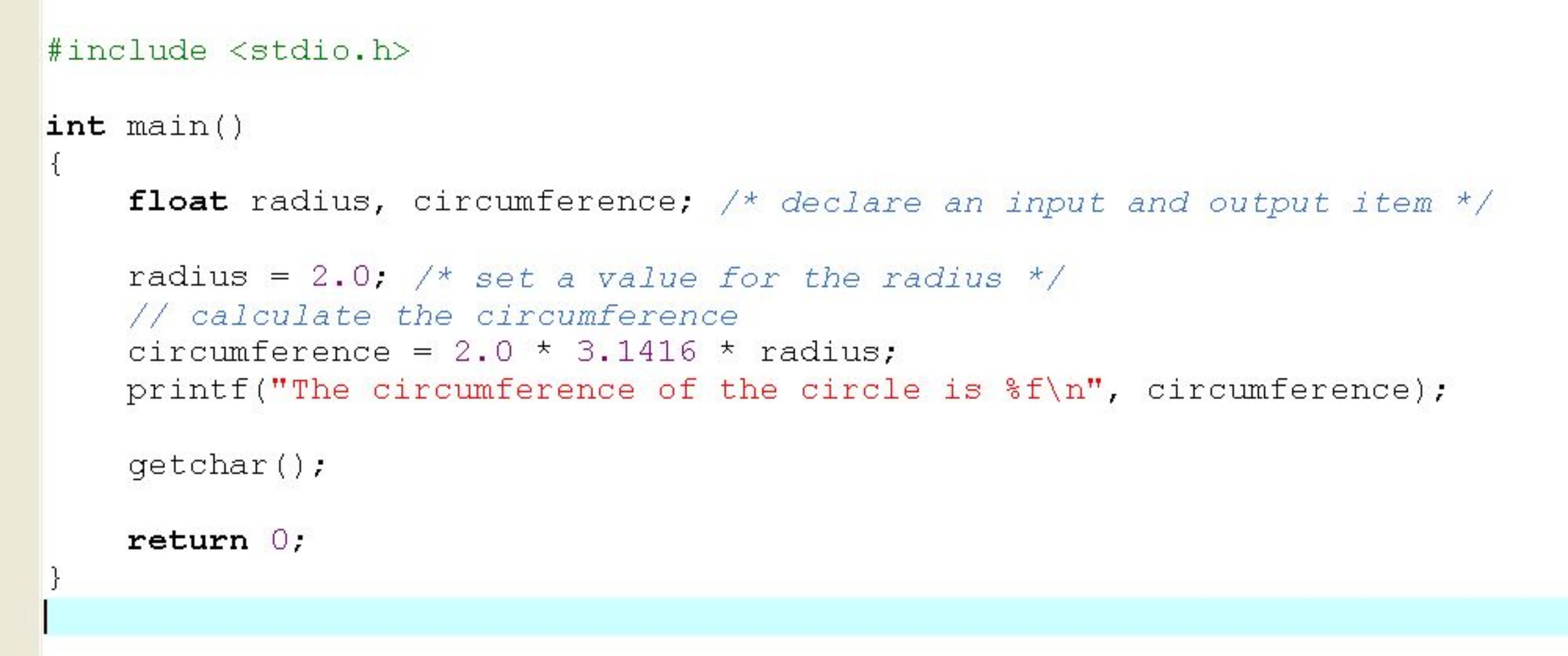
Function's arguments

`"Hello World"`

Figure 2.5 Passing a message to `printf()`

Comments - Explaining your Code

- Programmers can forgot their own code.
- Use comments to explain sections of code
- To create a comment in C, you surround the text with /* and then */ or using //



```
#include <stdio.h>

int main()
{
    float radius, circumference; /* declare an input and output item */

    radius = 2.0; /* set a value for the radius */
    // calculate the circumference
    circumference = 2.0 * 3.1416 * radius;
    printf("The circumference of the circle is %f\n", circumference);

    getchar();

    return 0;
}
```

Comments - Single Line

```
// single line
// comment
```

```
#include <stdio.h>
int main()
{
  // Print greeting
  printf("Hello World");
  return 0;
}

int test(){…}
```

Comments - Multiple Lines

```
/* multiple
   lines
   comment */
```

```
#include <stdio.h>
int main()
{
  /* Greeting from
     the earth */
  printf("Hello World");
  return 0;
}

int test(){…}
```

Comments - Multiple Lines

```
/* multiple-line comments */
```

CANNOT be nested e.g.

```
/* this comment is /* always */ invalid */
```

Variables

Variables

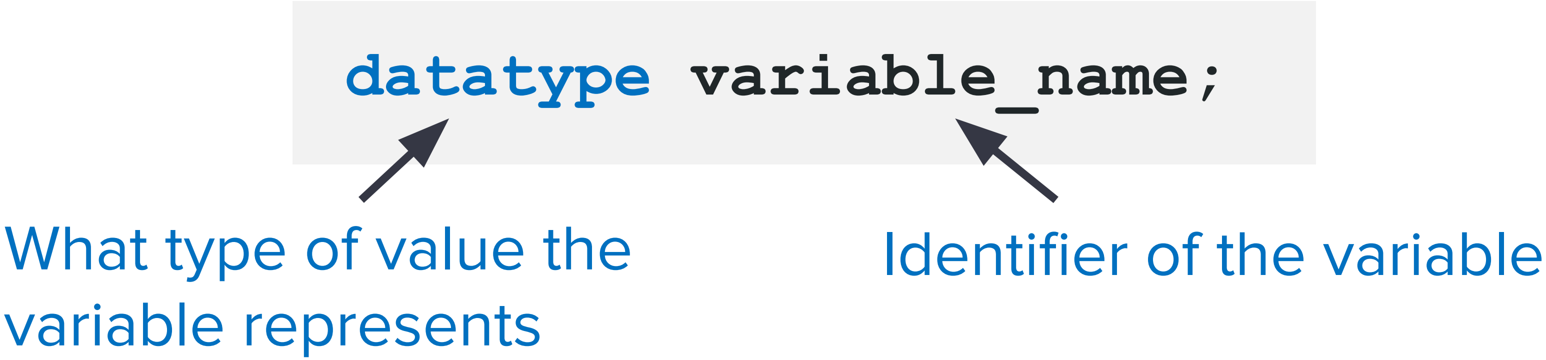
- A **variable** is a symbol or name for storing a value.
- Variables help us write flexible programs.

```
// area of a rectangle
area = width * height;
```

All rectangles of different widths and heights use the same code to compute the area.

Syntax of C variables

- Declaration



```
datatype variable_name;
```

What type of value the variable represents

Identifier of the variable

- You can declare many variables with the same data type

```
int width, height, area;
```

Note that: Detail of basic data type will be discussed later.

Variable Names

- A variable name is an **identifier**. You can name your variables to whatever you like, but keep three things in mind:
    - Format
    - Reserved words
    - Standard identifiers

1. Format

- Can be combination of any letter, digits, or underscores (_)
  - The first character of the identifier must be a letter or underscore
  - Only letters, digits, or underscores may follow the initial character
  - Blank spaces are not allowed
  - Cannot be a reserved words

- *Example*
  - Valid: `total,SUm,average,_x,y_,mark_1,x1`
  - Invalid: `1x,x+y`

2. Reserved words (keywords)

- Variable names **CANNOT BE** *reserved words*
- Some names are predefined by the programming language for only special purpose
- Using for other purposes will generate errors

```
auto   default  float  register struct  volatile
break  do       for    return   switch  while
case   double   goto   short    typedef char
else   if       signed union    const   enum
int    unsigned sizeof continue extern  long
static void
```

```
int case;
```
✗

3. Standard Identifiers

- Words predefined by C, as part of the standard library
- A programmer can redefine them

```
abs      fopen    isalpah  rand     strcpy
argc     free     malloc   rewind   strlen
argv     fseek    memcpy   scanf    tolower
calloc   gets     printf   sin      toupper
fclose   isacii   puts     strcat   ungetc
```

NOTE: Meaning of a Variable

- A variable name should relate to what the variable represents

```
// what are they?
a = ( b * c ) + d;

// how about now?
y = ( m * x ) + c;
```

Exercise: Are they valid?

Variable names:

```
check_items

4ab7

displayMessage123

hoursWorked

Total

e*6

rectangle area

TOTAL
```

Variables: Initialization and Assignment

- When a declaration statement provides an initial value, the variable is said to be initialized

```
int width = 10;
int height = 5, volume = 2;
int area;   // unknown
```

- Assignment: set value of a variable (**from right to left**)

```
int x, y;
x = -200;
y = x + 10;
```

Variables: Assignment (cont.)

- Operand on the right can be:
  - Values (numbers)
  - Variables
  - Any valid C expression

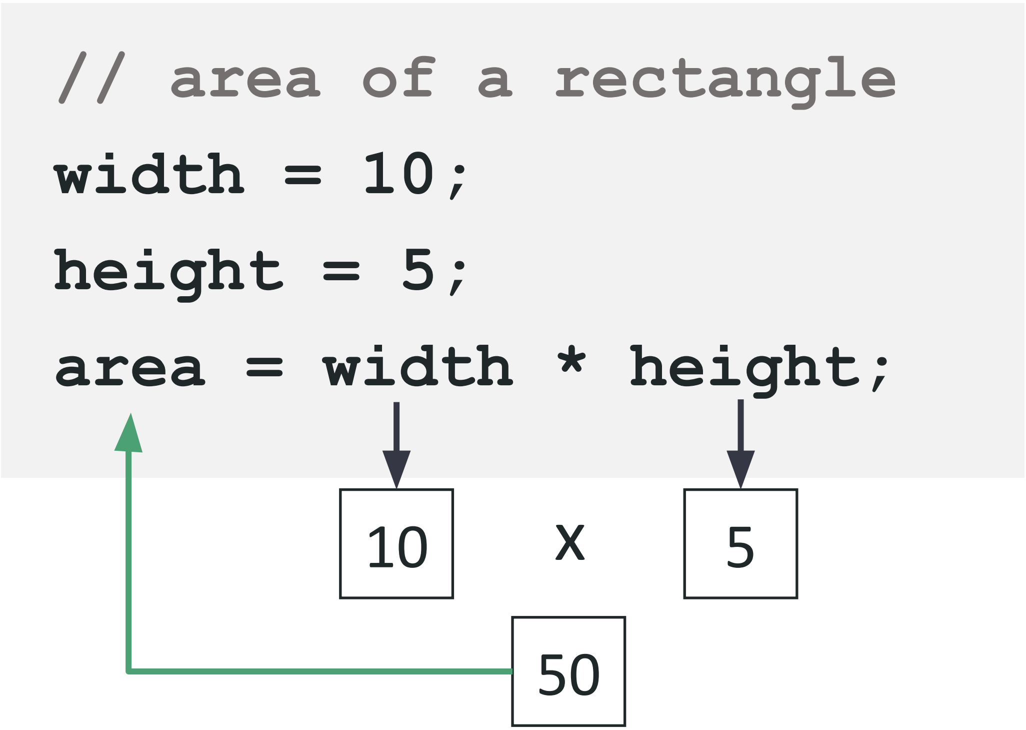
```
sum = 3 + 7;

diff = 15 - 6;

product = .05 * 14.6;

tally = count + 1;

newTotal = 18.3 + total;

slope = (y2 - y1) / (x2 - x1);
```

Expression on the right side is calculated first

All variables here need to be initialized

Variables: Assignment (cont.)

- When the program is executed, the variables are replaced with real data (values).



```
// area of a rectangle
width = 10;
height = 5;
area = width * height;
```

Variables: Assignment (cont.)

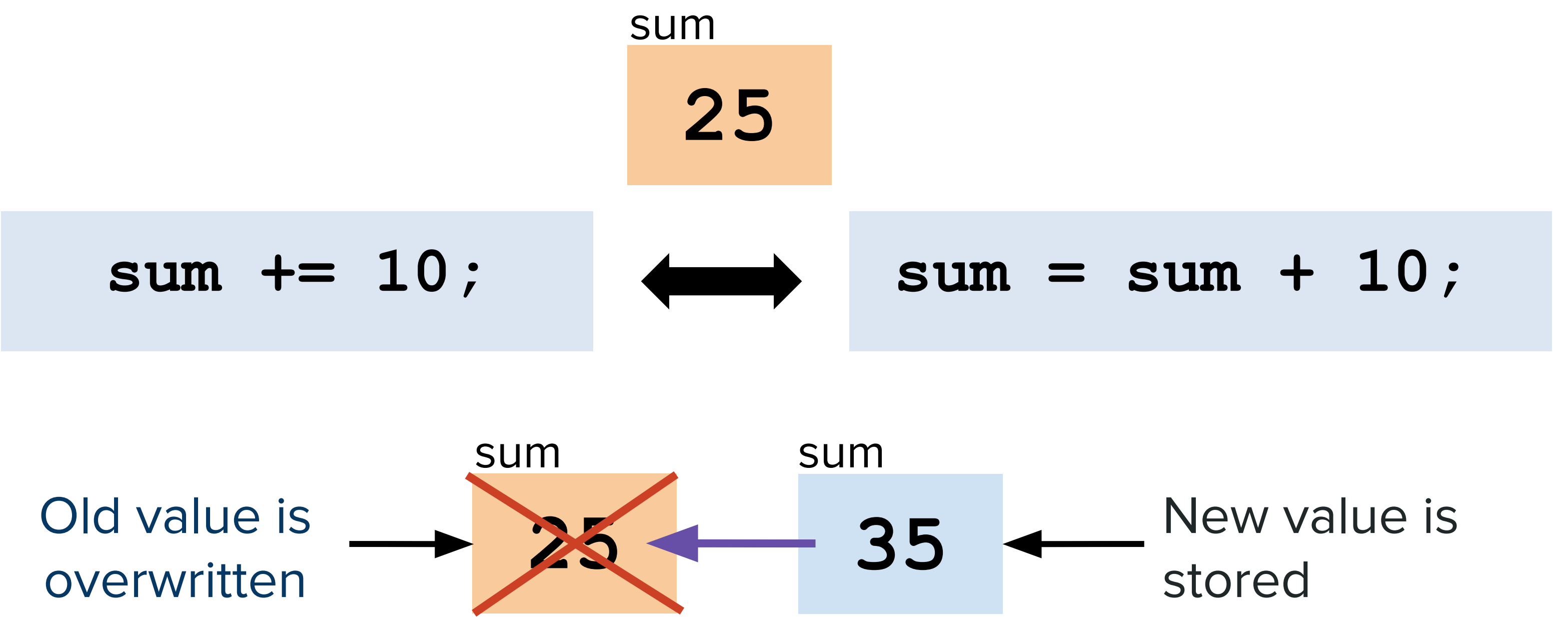
- Be careful!!!
- Only one variable on the left side

✗
```
amount + 1892 = 1000 + 10 * 5;
```

Variables: Assignment (cont.)

- Special assignment operators

```
+= -= *= /= %=
```



```
sum += 10;
```

```
sum = sum + 10;
```

Old value is overwritten

New value is stored

Exercise: What are the value?

- What is the value of `w`, `x`, `y`, and `z` ?

```
int w = 0, x, y, z;
x = 10;
x = x + 10;
y = 2 + 2;
```

Exercise: What are the value?

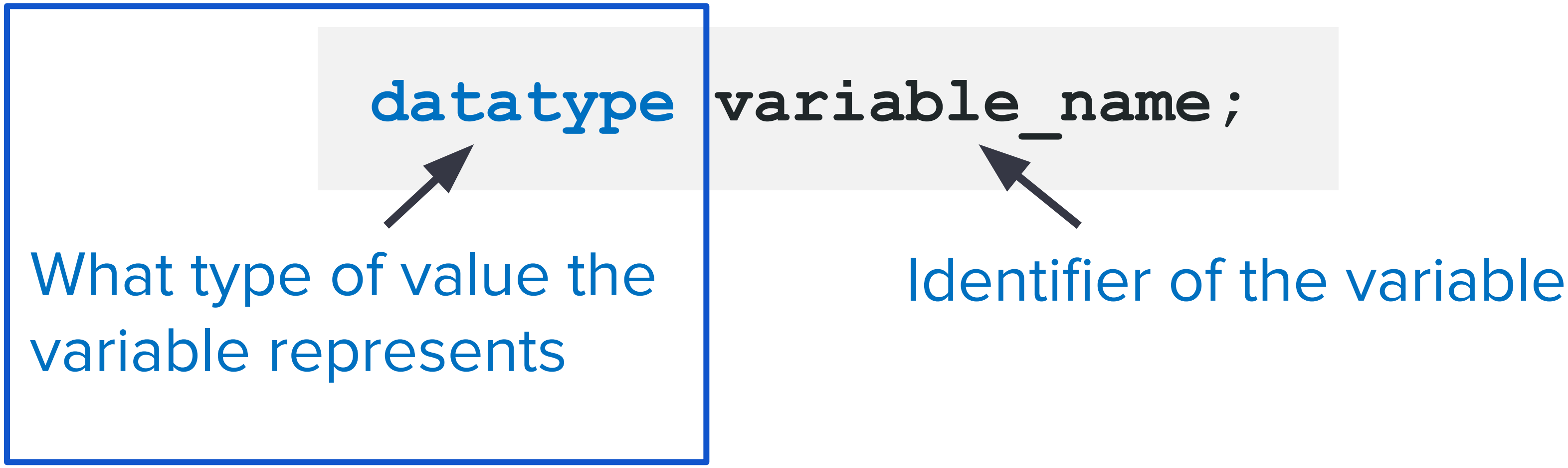
- What is the value of `x` and `y`?

```
int x, y;
int y = 100;
x = z + 210;
```

Data Type

Syntax of C variables: Data Type

- Declaration



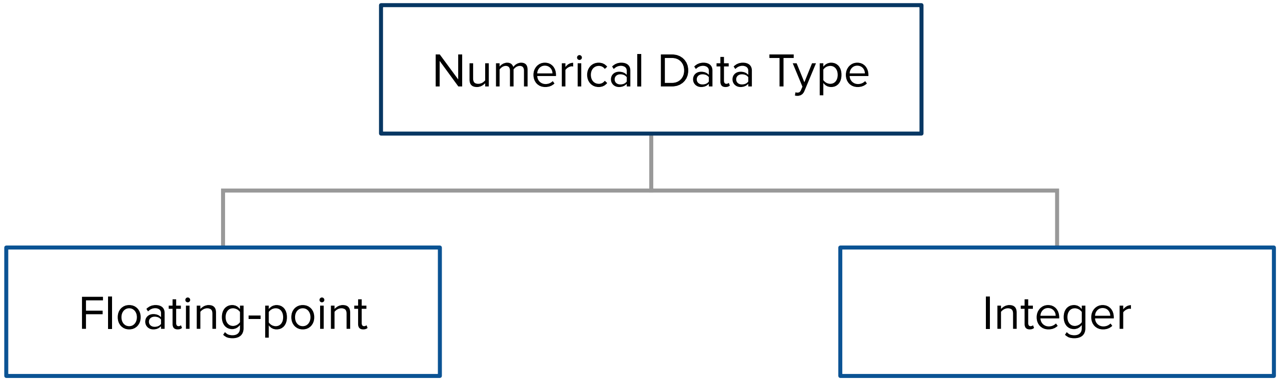
```
datatype variable_name;
```

What type of value the variable represents

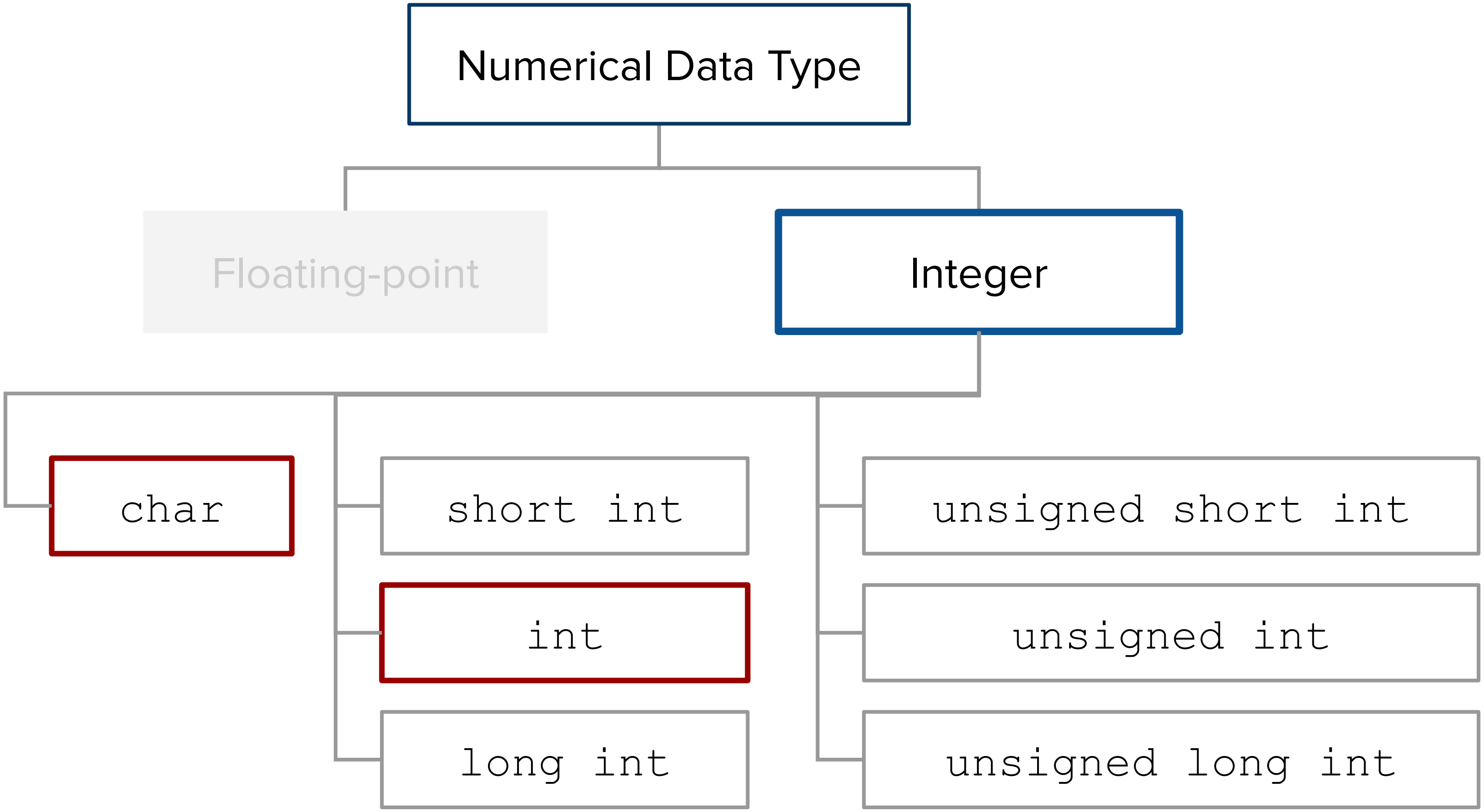
Identifier of the variable

What is Data Type?

- A kind of data that can be kept on the predefined variable.
- Built-in data types (primitive types) is provided by C
  - Numerical data types consists of Integer and Floating-point types



Integer Data Type



Integer Data Types: `int`

- `int` data type
  - Whole numbers and + or – signs
  - Values between -2,147,483,648 to 2,147,483,647

- *Example*
  - Valid integer constant:

    ```
    5    -10     +25    1000
    253  -26351 +36
    ```

  - Invalid integer constant:

    ```
    $255.62  2,523  3.  6,243,892
    1,492.89 +6.0
    ```

Integer Data Types: `char`

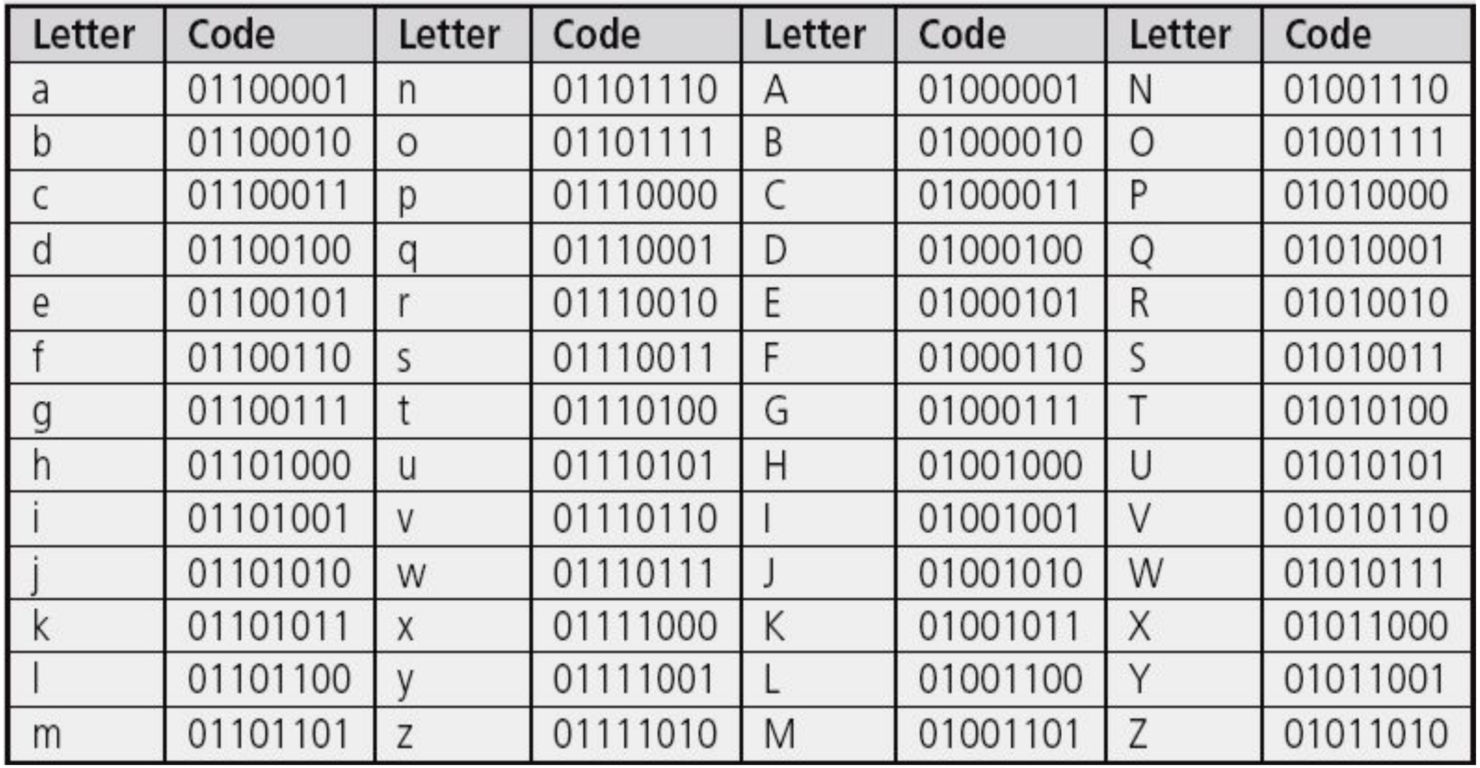
- **`char`** data type
  - Store individual character
  - Printable character: letters, digits, and special symbols

- *Example*
  - Letters: `'L'` `'o'` `'l'`
  - Digits: `'1'` `'0'` `'5'`
  - Special symbols: `'$'` `'#'` `','`

ASCII and ANSI code

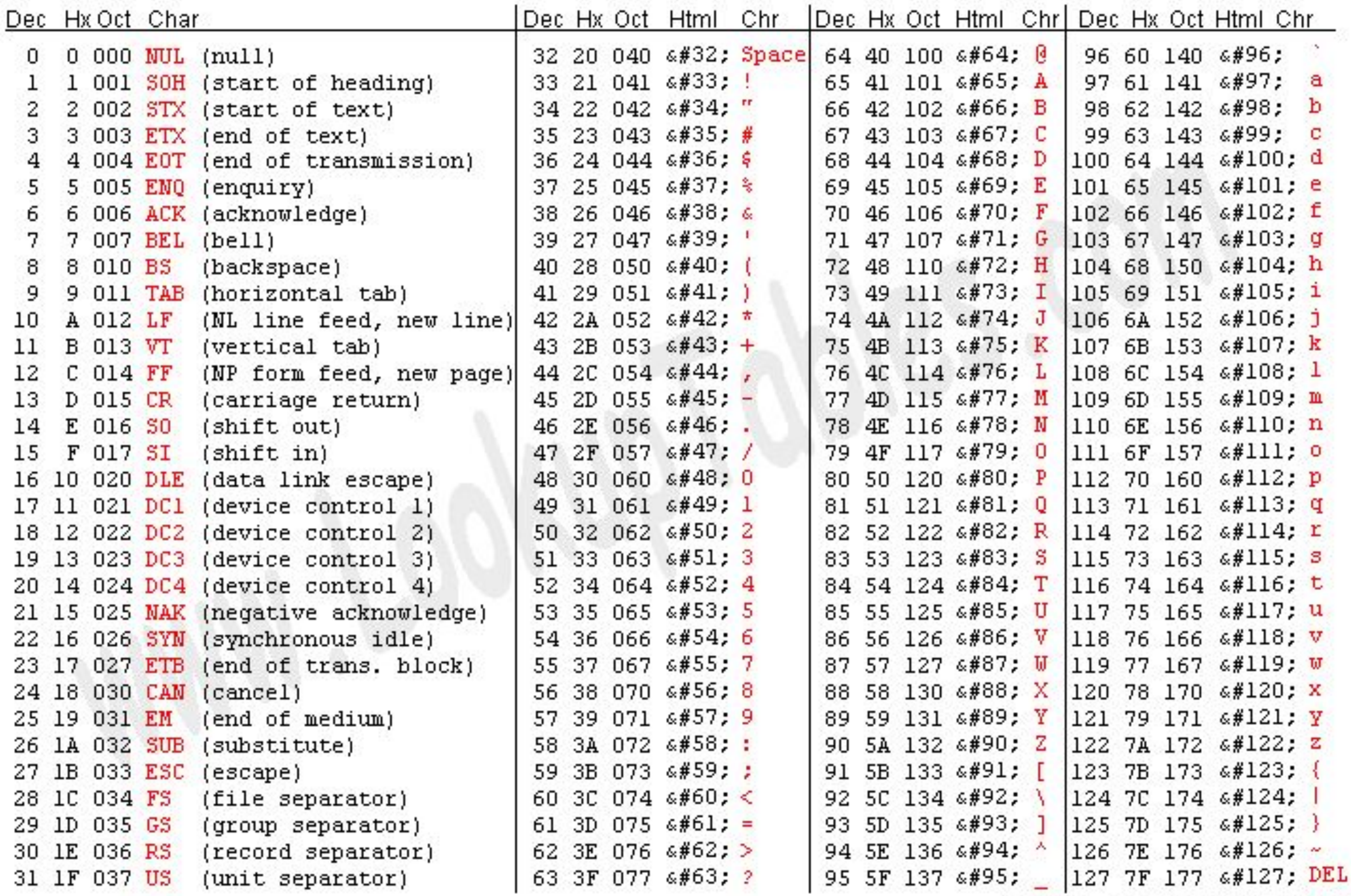
- Character encoding standards
  - ASCII: American Standard Code for Information Interchange (7 bits)
  - ANSI: American National Standards Institute (8 bits)

Table 2.4 ASCII and ANSI Letter Codes



Letter	Code	Letter	Code	Letter	Code	Letter	Code
a	01100001	n	01101110	A	01000001	N	01001110
b	01100010	o	01101111	B	01000010	O	01001111
c	01100011	p	01110000	C	01000011	P	01010000
d	01100100	q	01110001	D	01000100	Q	01010001
e	01100101	r	01110010	E	01000101	R	01010010
f	01100110	s	01110011	F	01000110	S	01010011
g	01100111	t	01110100	G	01000111	T	01010100
h	01101000	u	01110101	H	01001000	U	01010101
i	01101001	v	01110110	I	01001001	V	01010110
j	01101010	w	01110111	J	01001010	W	01010111
k	01101011	x	01111000	K	01001011	X	01011000
l	01101100	y	01111001	L	01001100	Y	01011001
m	01101101	z	01111010	M	01001101	Z	01011010

Decimal ASCII code value



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

Source: www.LookupTables.com

The Escape Character

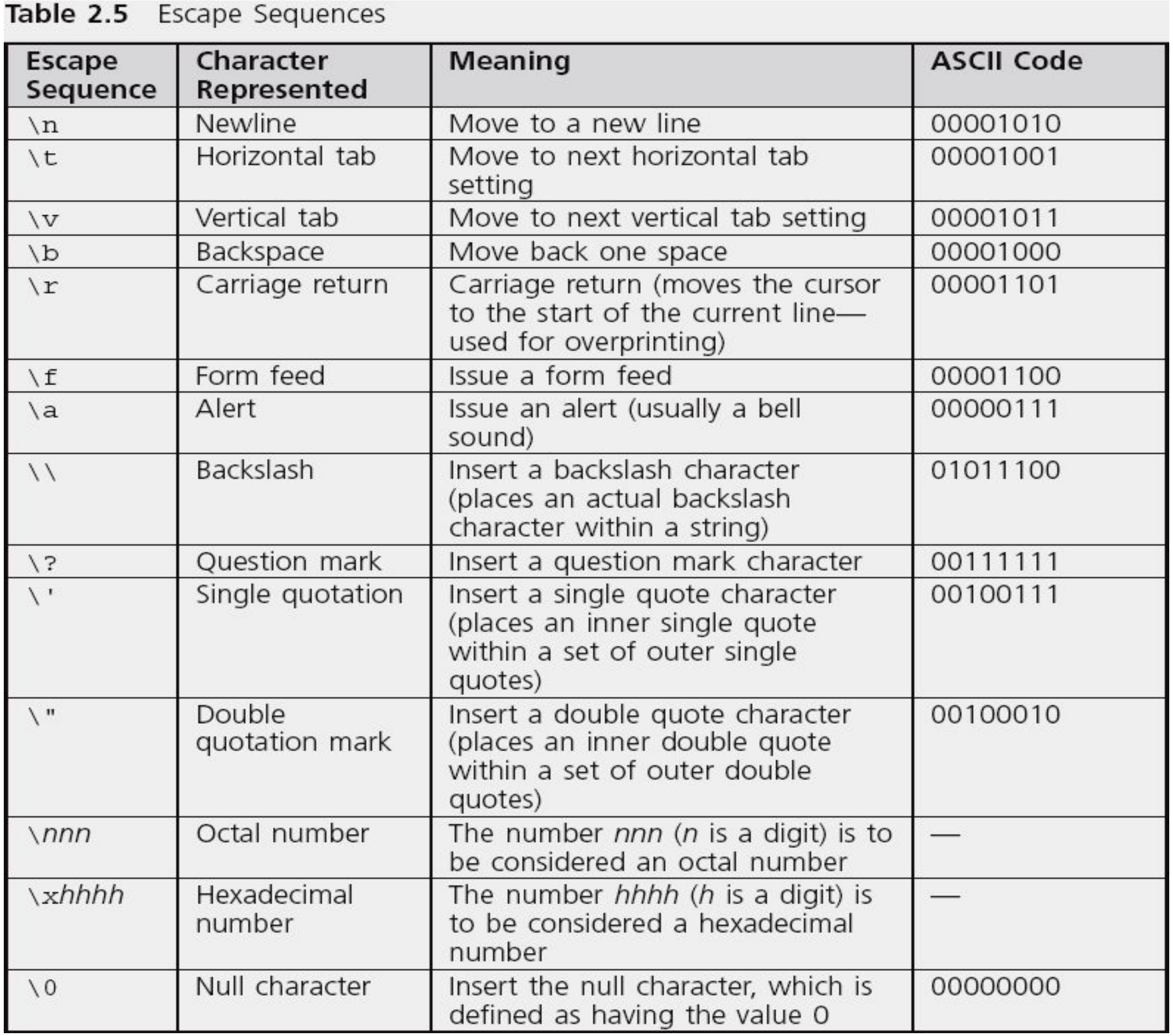
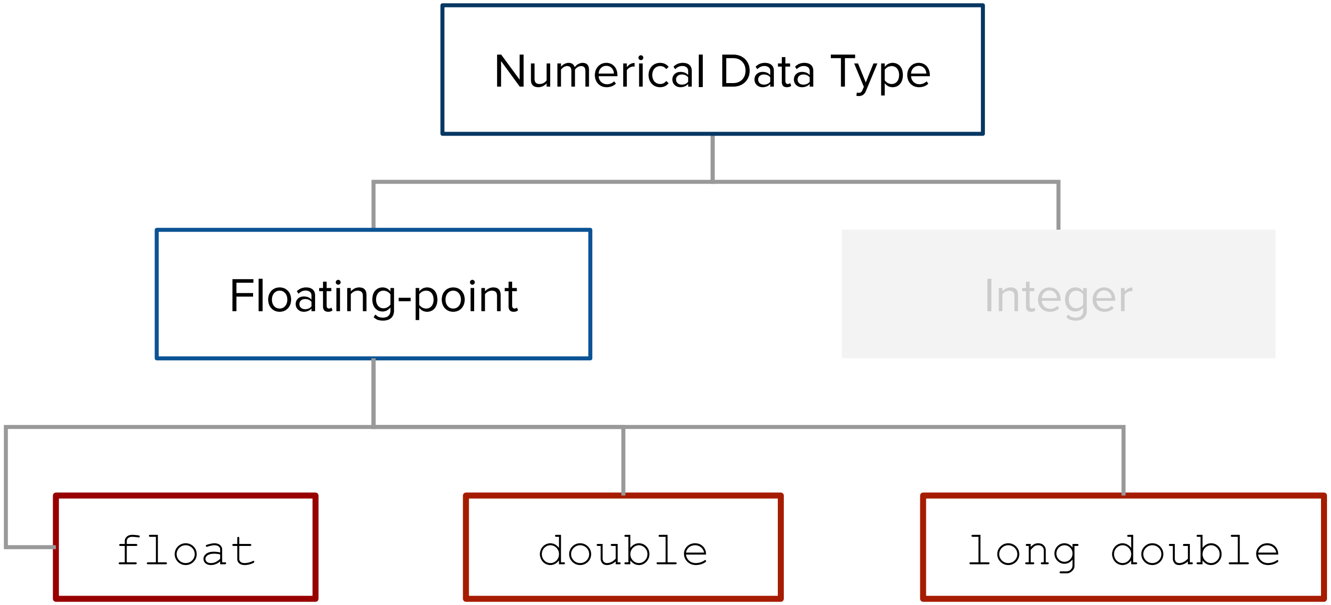


Table 2.5 Escape Sequences

Escape Sequence	Character Represented	Meaning	ASCII Code
`\n`	Newline	Move to a new line	00001010
`\t`	Horizontal tab	Move to next horizontal tab setting	00001001
`\v`	Vertical tab	Move to next vertical tab setting	00001011
`\b`	Backspace	Move back one space	00001000
`\r`	Carriage return	Carriage return (moves the cursor to the start of the current line—used for overprinting)	00001101
`\f`	Form feed	Issue a form feed	00001100
`\a`	Alert	Issue an alert (usually a bell sound)	00000111
`\\`	Backslash	Insert a backslash character (places an actual backslash character within a string)	01011100
`\?`	Question mark	Insert a question mark character	00111111
`\'`	Single quotation	Insert a single quote character (places an inner single quote within a set of outer single quotes)	00100111
`\"`	Double quotation mark	Insert a double quote character (places an inner double quote within a set of outer double quotes)	00100010
\nnn	Octal number	The number *nnn* (*n* is a digit) is to be considered an octal number	—
\\x*hhhh*	Hexadecimal number	The number *hhhh* (*h* is a digit) is to be considered a hexadecimal number	—
`\0`	Null character	Insert the null character, which is defined as having the value 0	00000000

Floating-point Data Type



Floating-point Data Types

- Also called “real number”
- Can be number zero or any positive or negative number that contains a decimal point

- *Example*
    - Valid floating-point constant:

      +10.6255   5.   -6.2   3251.92

      0.0   0.33   -6.67   +2.
    - Invalid floating-point constant:

      5,326.25   24   123   6,459 $10.29

Floating-point Data Types

- Three types of floating-point:

1. `float  (9.234f)`
2. `double (9.234)`
3. `long double (9.234L)`

Single-precision

Double-precision

Basic In & Out … Get a value and display

Basic input & output

- **Input**: Use `scanf` for getting value from users and assign the obtained value to a given variable
- **Output**: Use `printf` for displaying the value of a given variable
- Both functions needs a conversion control sequence

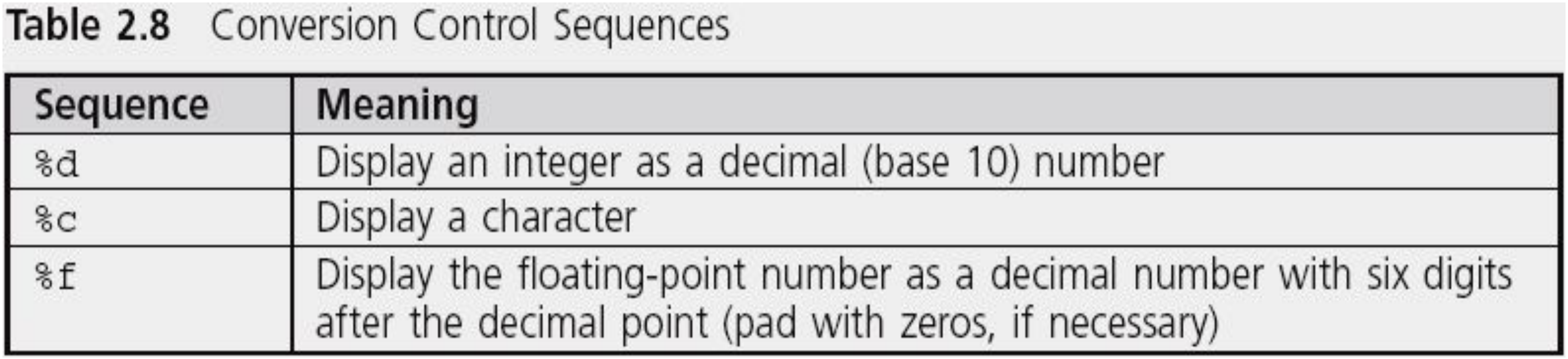


Table 2.8 Conversion Control Sequences

Sequence	Meaning
`%d`	Display an integer as a decimal (base 10) number
`%c`	Display a character
`%f`	Display the floating-point number as a decimal number with six digits after the decimal point (pad with zeros, if necessary)

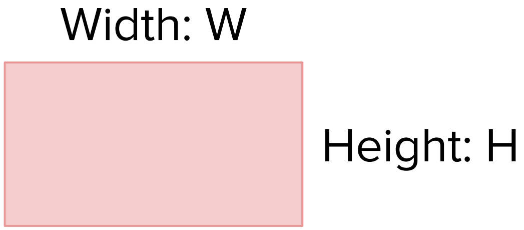
Data Type `int`

Conversion Control Sequence

- **Input**: `scanf("%d",&variable_inttype);`

  Example

  - `scanf("%d",&width);`
  - `scanf("%d",&height);`



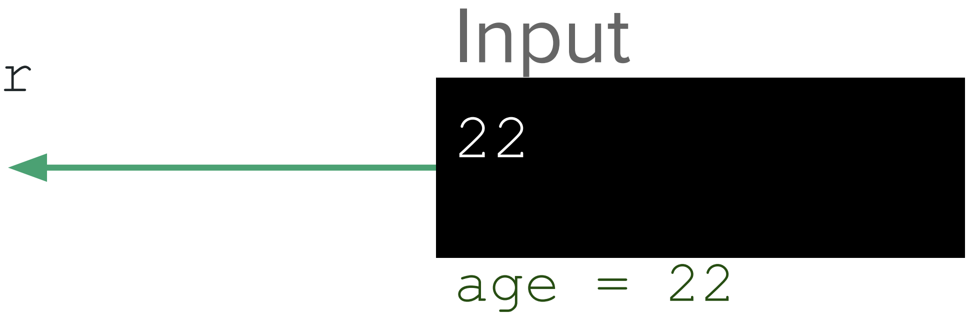
- **Output**: `printf("%d", variable_inttype);`

  Example

  - `printf("%d", area);`
  - `printf("%d", 7+8);`

Example

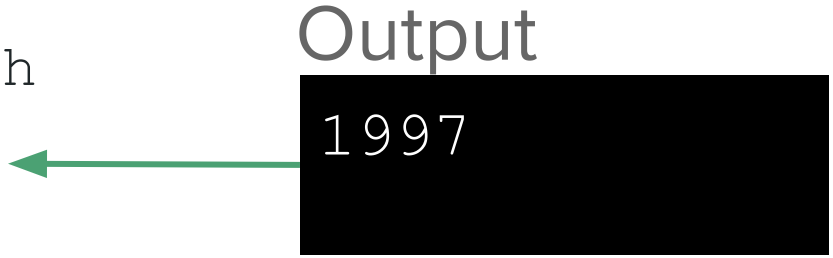
```c
#include <stdio.h>
int main() {
    int age;
    //Input from a user
    scanf("%d",&age);


    //Display year of birth
    printf("%d",2019-age);
    return 0;
}
```



Input

```
22
```

age = 22



Output

```
1997
```

Data Type `char`

- **Input**: `scanf("%c",&variable_chartype);`

  Example

  - Receive an input color code: R, G, B

    `scanf("%c",&color);`

- **Output**: `printf("%c",variable_chartype);`

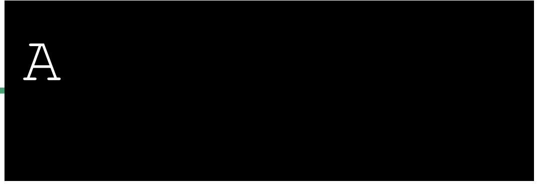
  Example

  - `printf("%c", color);`
  - `printf("%c", '$'); //Character value of '$'`
  - `printf("%d", '$'); //Decimal value of '$'`

Example

```c
#include <stdio.h>
int main() {
    char alphabet;
    scanf("%c",&alphabet);

    /* Display character and
    ASCII code of an alphabet */
    printf("%c %d\n", alphabet, alphabet);
    return 0;
}
```

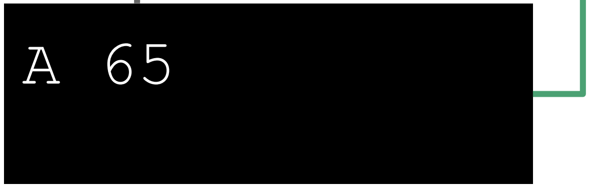
Input



```
A
```

alphabet = 'A'

Output



```
A 65
```

Data Type `float`

- **Input**: `scanf("%f",&variable_floattype);`

  Example

  - `scanf("%f",&money);`
  - `scanf("%f",&num);`

- **Output**: `printf("%f",variable_floattype);`

  Example

  - `printf("%.2f", money); //Two decimal points`
  - `printf("%f", average);`

Example

```
#include <stdio.h>
int main() {
   float num1, num2;
   scanf("%f %f",&num1, &num2);

   //Display sum of two numbers
   printf("%f\n", num1+num2);
   printf("%.2f\n", num1+num2);

   return 0;
}
```

Input

```
2.5 1
```

```
num1 = 2.500000
num2 = 1.000000
```

Output

```
3.500000
3.50
```

Operators

Operators

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Increment, Decrement operators
5. Assignment operators

1 Arithmetic Operations

Operation	Operator	Type	Operand	Result
Addition	+	Binary	Both are integers	Integer
			One operand is a floating-point number	Floating-point number
Subtraction	-	Binary	Both are integers	Integer
			One operand is a floating-point number	Floating-point number
Multiplication	*	Binary	Both are integers	Integer
			One operand is a floating-point number	Floating-point number
Division	/	Binary	Both are integers	Integer
			One operand is a floating-point number	Floating-point
Modulus	%	Binary	Both are integers	Integer
Negation	-	Unary	Integer or floating-point	Same as operand

Binary: Require two operands
Unary: Require one operand

Example: Arithmetic Operators

- Addition
  - ✔ `x = 7 + 3;` // constants
  - ✔ `x = y + z;` // variables
  - ✔ `x = y + z + 1;` // both

- Subtraction
  - ✔ `x = 7 - 3;` // constants
  - ✔ `x = y - z;` // variables
  - ✔ `x = y - z - 1;` // both

An operand can be either a literal value or a variable that has a value associated with it

Example: Arithmetic Operators

- Modulus

```
✔   int x = 5%2;                       //x = ?
✔   int y = 10; int x = y%3;          //x = ?
✔   int y = 5; int x = y%10;         //x = ?
```

- Negative

```
✔   int x = -5;
✔   int y = 3; int x = -y;
```

2 Relational Operators

- Compare two operands to produce a Boolean result
- **True**: non-zero value (i.e. 1)
- **False**: 0

Operator	Meaning	Example

>	Greater than	3 > 2;
		2 > 3;

>=	Greater than or equal to	3 >= 3;
		2.9 >= 3;

<	Less than	3 < 2;
		2 < 3;

<=	Less than or equal to	3 <= 3;
		3.1 <= 3;

==	Equal to	3 == 3;
		2 == 3;

!=	Not equal to	3 != 3;
		2 != 3;

True or False ?

Note that

- "==" equality operator is different from the "=", **assignment operator**
- the "==" operator on float variables is tricky because of finite precision

3 Logical Operators

True or False ?

<table>
<tr><th>Operator</th><th>Meaning</th><th>Example</th></tr>
<tr><td>&&</td><td>AND</td><td>(25/5 == 5) && (2+3 == 5);
(3*2 == 6) && (2+3 == 6);</td></tr>
<tr><td>||</td><td>OR</td><td>(25/5 == 5) || (2+3 == 5);
(3*2 == 6) || (2+3 == 6);</td></tr>
<tr><td>!</td><td>NOT</td><td>!(3*2 == 6);
!(2+3 == 6);</td></tr>
</table>

4 Increment, Decrement Operators

- **Postfix**: increment the value **after** using it.
  - `x++;` means `x = x+1;`
  - `x--;` means `x = x-1;`
  - `y = x++;` means `y = x;`
    `x = x+1;`
  - `y = x--;` means `y = x;`
    `x = x-1;`

```
int i;
i = 6;
printf("%d ", i++);
printf("%d ", i);
```

```
int i;
i = 6;
printf("%d ", i--);
printf("%d ", i);
```

- **Prefix**: increment the value **before** using it.
  - `++x;` means `x = x+1;`
  - `--x;` means `x = x-1;`
  - `y = ++x;` means `x = x+1;`
    `y = x;`
  - `y = --x;` means `x = x-1;`
    `y = x;`

```
int i;
i = 6;
printf("%d ", ++i);
printf("%d ", i);
```

```
int i;
i = 6;
printf("%d ", --i);
printf("%d ", i);
```

5 Assignment Operators

- Assignment operation
  - `x = x+1;`
  - `x = x-1;`
  - `x = x*3;`
  - `x = x/3;`
  - `x = x%3;`
- Compact assignment operation
  - `x+=1;`
  - `x-=1;`
  - `x*=3;`
  - `x/=3;`
  - `x%=3;`

Mathematical Library Functions

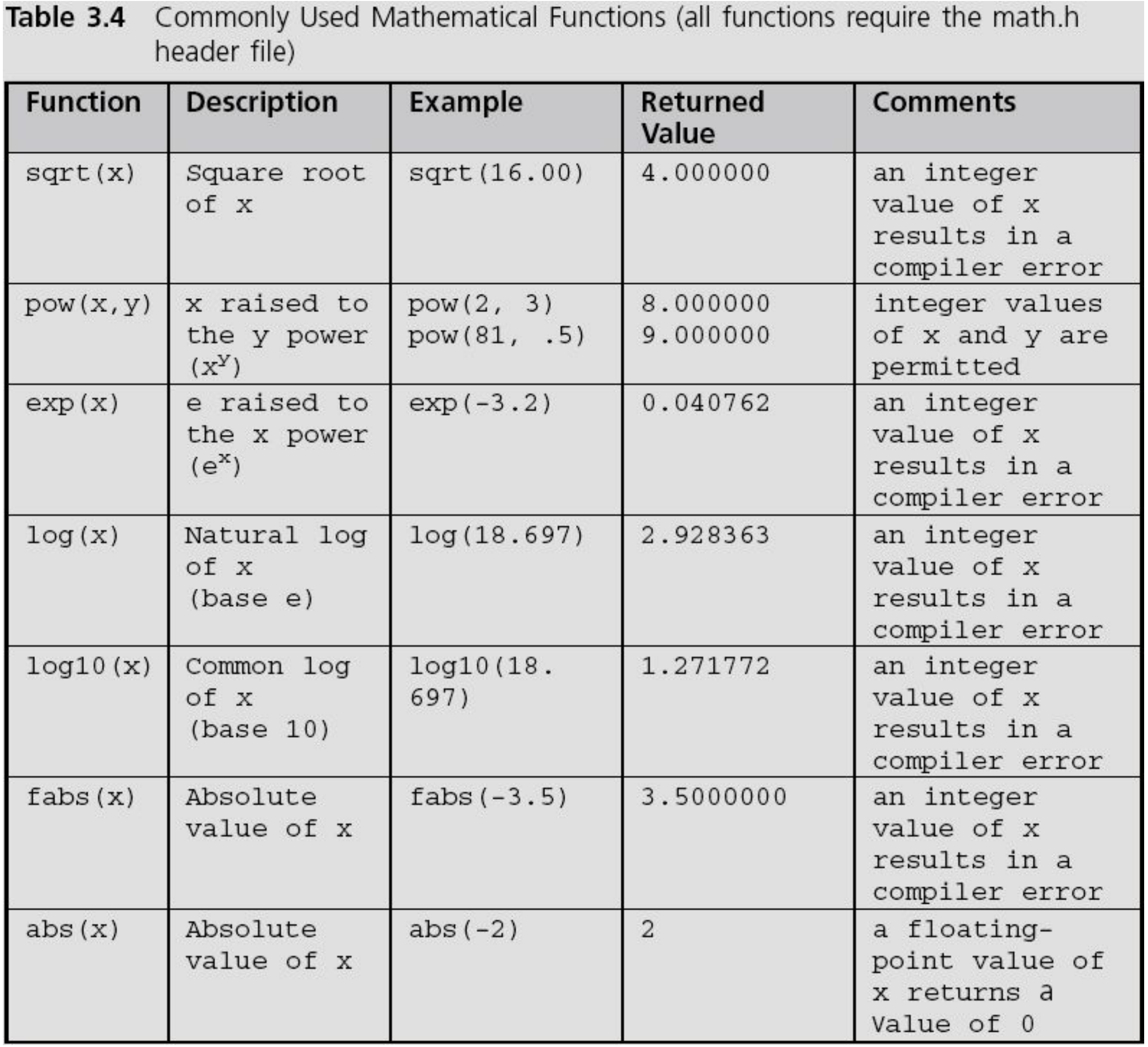


Table 3.4 Commonly Used Mathematical Functions (all functions require the math.h header file)

Function	Description	Example	Returned Value	Comments
sqrt(x)	Square root of x	sqrt(16.00)	4.000000	an integer value of x results in a compiler error

pow(x,y)	x raised to the y power (x^y)	pow(2, 3)
		pow(81, .5)
			8.000000
			9.000000
				integer values of x and y are permitted

exp(x)	e raised to the x power (e^x)	exp(-3.2)	0.040762	an integer value of x results in a compiler error
log(x)	Natural log of x (base e)	log(18.697)	2.928363	an integer value of x results in a compiler error
log10(x)	Common log of x (base 10)	log10(18. 697)	1.271772	an integer value of x results in a compiler error
fabs(x)	Absolute value of x	fabs(-3.5)	3.5000000	an integer value of x results in a compiler error
abs(x)	Absolute value of x	abs(-2)	2	a floating-point value of x returns a Value of 0

Note: don't forget to `#include <math.h>`

Expression

Expression: any combination of operators and operands that can be evaluated to yield a value

- **Integer expression:** contains only integer operands; the result is an integer
  - integer + integer ➔ integer

- **Floating-point expression:** contains only floating-point operands; the result is a double-precision
  - real (floating-point) + real ➔ double-precision

- In a **mixed-mode expression** the data type of each operation is determined by the following rules:
  - If both operands are integers, result is an integer
  - If one operand is real, result is double-precision

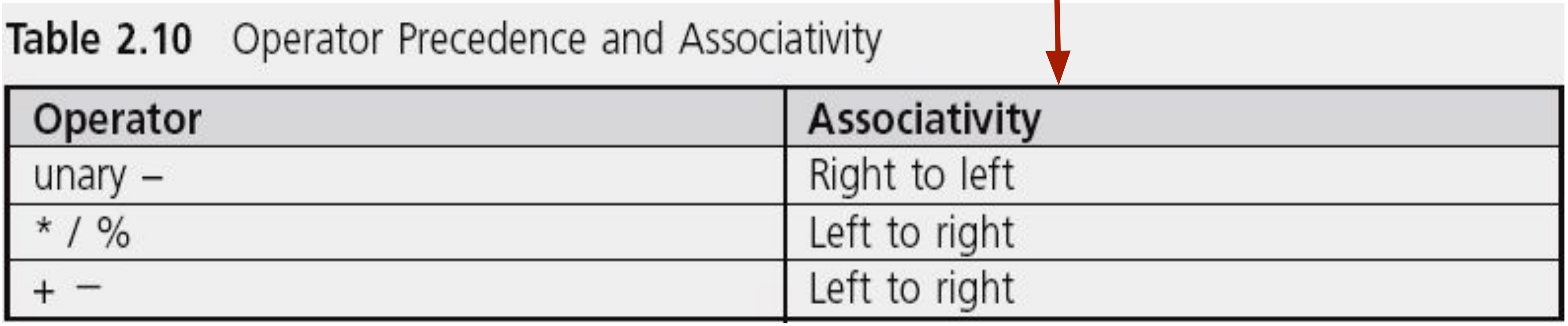
Operator Precedence and Associativity (cont.)

- Three levels of **precedence**:
  - All negations are done first
  - Multiplication, division, and modulus operations are computed next; expressions containing more than one of these operators are evaluated from left to right as each operator is encountered
  - Addition and subtraction are computed last; expressions containing more than one addition or subtraction are evaluated from left to right as each operator is encountered

Operator Precedence and Associativity

- Two binary arithmetic operator symbols must never be placed side by side
- Parentheses may be used to form groupings and expressions in parentheses are evaluated first
- Parentheses may be enclosed by other parentheses
- Parentheses cannot be used to indicate multiplication

Operator Precedence



Level of precedence

Table 2.10 Operator Precedence and Associativity

Operator	Associativity
unary –	Right to left
* / %	Left to right
+ −	Left to right

Example: Operator Precedence

- Find the result of `6 + 4 / 2 + 3`

```
6 + 2 + 3
   8 + 3
   11
```

- Find the result of `8 + 5 * 7 % 2 * 4`

```
   8 + 35 % 2 * 4
   8 + 1 * 4
   8 + 4
   12
```

Suggestion: If you’re not sure, always use `()` to ensure your result

Data Type Conversion

- Sometimes, we need to perform operation between different data types. Data type conversion is needed.

- There are two types of conversion in C
    - Implicit type conversion : automatic conversion of data type in order to evaluate the expression.
    - Explicit type conversion: manually decide what type we want to convert the expression.

Implicit Type Conversion

- Example: Implicitly converted to a floating-points type

```
int result = 4;

float result2 = result;

printf("%f", result2);
```

Output

```
4.000000
```

Explicit Type Conversion (Casts)

- “Cast” `double` to `int`. Decimal point is truncated

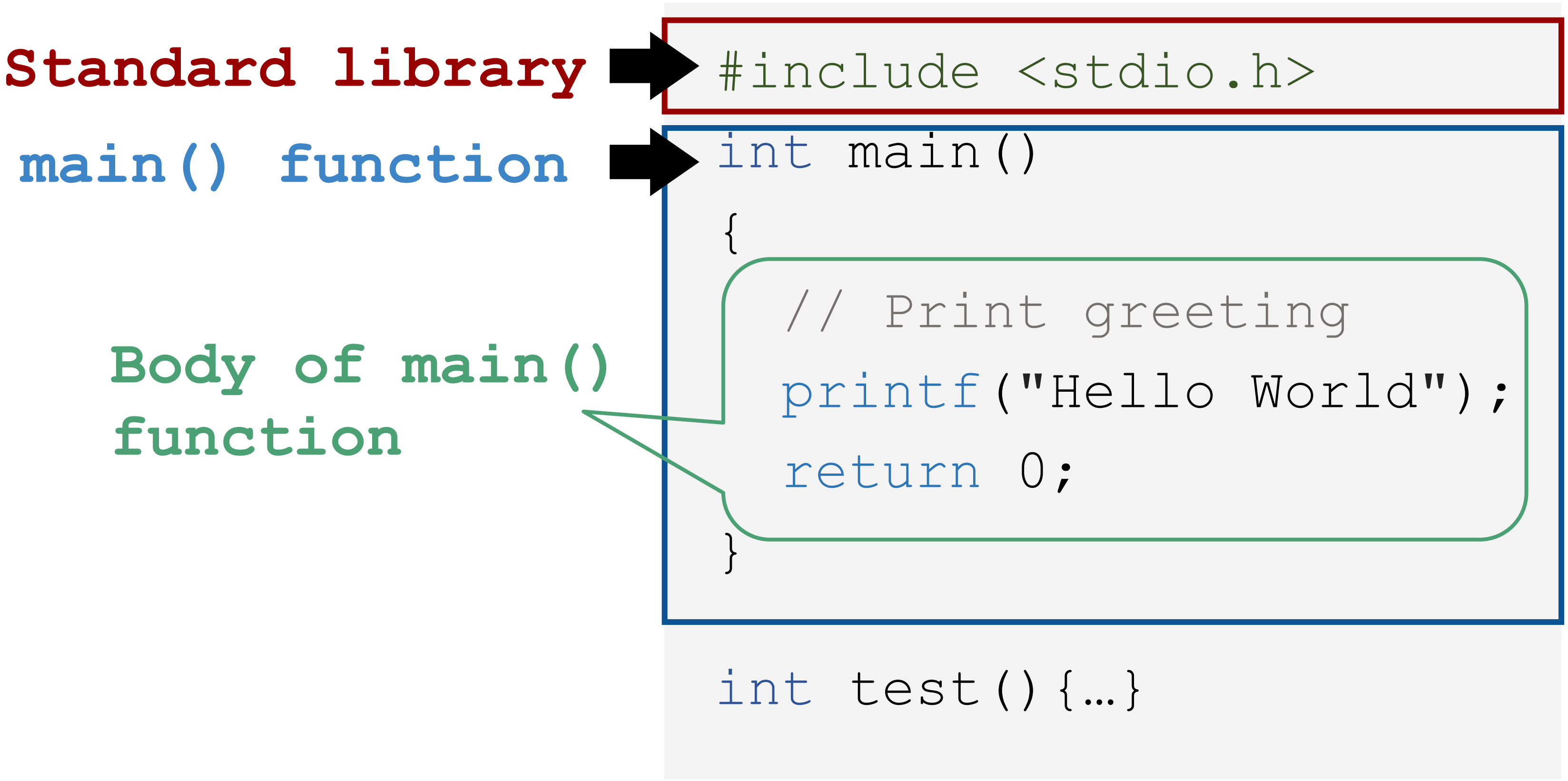
```
(datatype) expression;
```

```
double sum = 5.5 + 12.5;
printf("%f\n", sum);
printf("%d", (int)sum);
```

Output

```
18.000000
18
```

Summary



Standard library

main() function

Body of main() function

```
#include <stdio.h>
int main()
{
    // Print greeting
    printf("Hello World");
    return 0;
}

int test(){…}
```

Summary

- Variables
  - Declaration and Initialization
  - Variable names
- Data types
  - Integer: `int`, `char`
  - Floating-point: `float`, `double`, `long double`
- Arithmetic Operators: `+`, `-`, `*`, `/`, `%`
- Relational Operators: `>=`, `==`, `<`, `!=`, etc.
- Logical Operators: `&&`, `||`, `!`
- Operator Precedence
- Type Conversion

Errors, Debugging and Backup

Bugs and Errors

bug *noun* (COMPUTER PROBLEM)

a mistake or problem in a computer program.

Exercise

How many **bugs** you spot in this program?

```
include <stdio.h>

int main(){
   int x; printf("Hello ");
   printf("World\n"):
   return x
}
```

Errors and Debugging

How to get it right:

- Do not Panic when you face compile errors
- Calmly look at the error message. It usually points you to the right line of code.
- Narrow down the problem by using comments and `printf` statements to check the value.
- Try to check and solve one by one

Common Programming Errors

- Omitting the parentheses after main

    `main` → `main()`

- Omitting or incorrectly typing the opening brace { that signifies the start of a function body
- Omitting or incorrectly typing the closing brace } that signifies the end of a function
- Misspelling the name of a function

    `print()` → `printf()`

Common Programming Errors (cont.)

- Forgetting to close the message to `printf()` with a double quote (`"   "`) symbol
- Omitting the semicolon (;) at the end of each statement
- Forgetting the `\n` to indicate a new line

Demo

```c
#include <stdio.h>
int main() {
  int a = 3,
  int b = 4, c = 5;
  average = a + b + c / 3;
  printf("Average: %d", average);
  return 0;
}
```

Let’s go the the lab