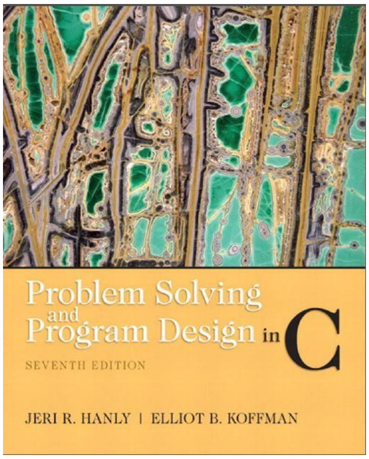
جامعة بيرزيت
BIRZEIT UNIVERSITY

Overview of C – Part 1

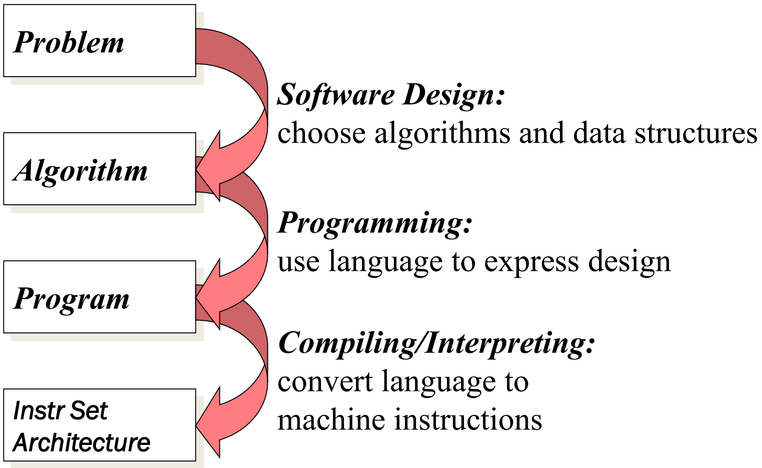
Computer Science Department


Problem Solving and Program Design in C
SEVENTH EDITION
JERI R. HANLY | ELLIOT B. KOFFMAN

Reference: *Problem Solving & Program Design in C*

How do we solve a problem using a computer?

A systematic sequence of transformations between layers of abstraction.



Problem

Algorithm

Program

Instr Set Architecture

Software Design:
choose algorithms and data structures

Programming:
use language to express design

Compiling/Interpreting:
convert language to
machine instructions

From a High-Level Program to an Executable File

a) Create file containing the program with a text editor.
b) Run preprocessor to convert source file directives to source code program statements.
c) Run compiler to convert source program into machine instructions.
d) Run linker to connect hardware-specific code to machine instructions, producing an executable file.

- Steps b–d are often performed by a single command or button click.
- Errors detected at any step will prevent execution of following steps.

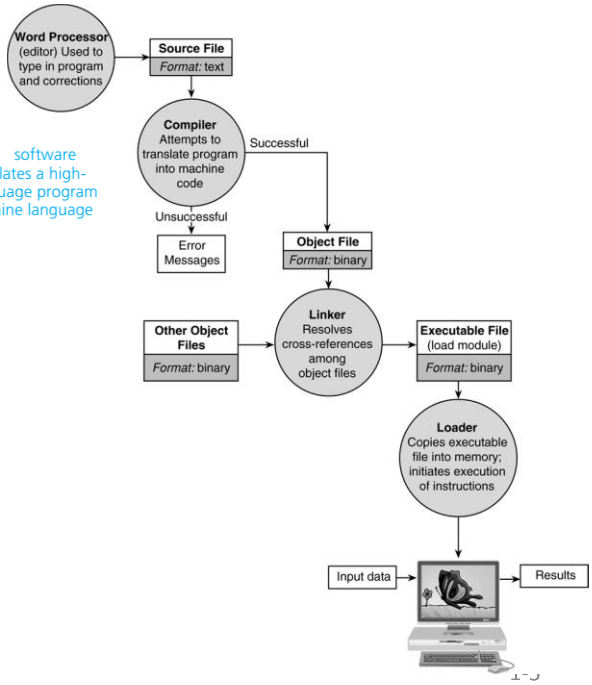
source file file containing a program written in a high-level language; the input for a compiler

syntax grammar rules of a programming language

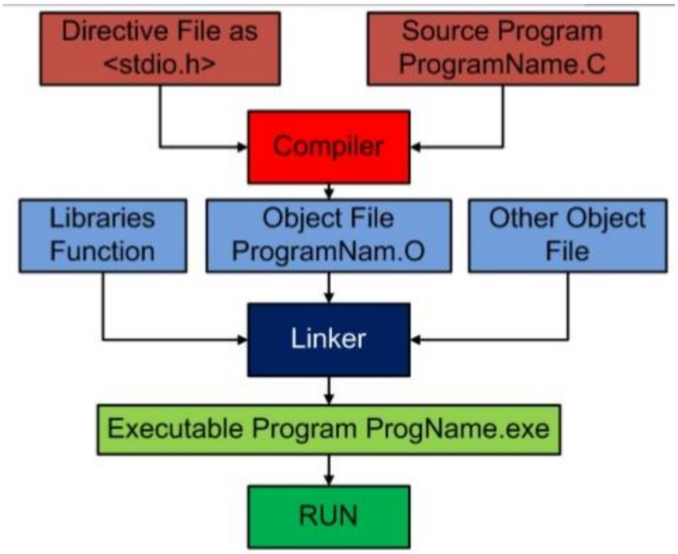
object file file of machine language instructions that is the output of a compiler

linker software that combines object files and resolves crossreferences to create an executable machine language program

compiler software that translates a high-level language program into machine language


Word Processor (editor) Used to type in program and corrections
Source File
Format: text
Compiler Attempts to translate program into machine code
Successful
Unsuccessful
Error Messages
Object File
Format: binary
Other Object Files
Format: binary
Linker Resolves cross-references among object files
Executable File (load module)
Format: binary
Loader Copies executable file into memory; initiates execution of instructions
Input data
Results

Steps of Obtaining an Executable Program


Directive File as <stdio.h>

Source Program ProgramName.C

Compiler

Libraries Function

Object File ProgramNam.O

Other Object File

Linker

Executable Program ProgName.exe

RUN

Motivation

```
//C program for area of circle   Comment
 #include <stdio.h> // standard header file (contains printf and scanf )
#define PI  3.141   //we use define for creating constant
int main()                // int, float , and return   are   reserved words
{
    float r, a;               // r, a are variables
    printf(“Please enter the radius: ");
    scanf("%f", &r);
    a = PI * r * r;          //= , *,{, }    special symbols
    printf("%f\n", a);
    return 0;
}
```

Preprocessor Directives

- #include <stdio.h>
  - notify the preprocessor that some names used in the program are found in <stdio.h>
- #define
  - using only data values that never change should be given names

File **stdio.h** Content

Input/Output functions

fprintf	Formatted File Write
fscanf	Formatted File Read
printf	Formatted Write
scanf	Formatted Read

File Operation functions

fclose	Close File
fflush	Flush File Buffer
fopen	Open File

Character Input/Output functions

fgetc	Read Character from File
fgets	Read String from File
fputc	Write Character to File
fputs	Write String to File
getc	Read Characters from File
getchar	Read Character
gets	Read String
putc	Write Character to File
putchar	Write Character
puts	Write String

....

Other Header Files

<assert.h>	Diagnostics Functions
<ctype.h>	Character Handling Functions
<locale.h>	Localization Functions
<math.h>	Mathematics Functions
<setjmp.h>	Nonlocal Jump Functions
<signal.h>	Signal Handling Functions
<stdarg.h>	Variable Argument List Functions
<stdio.h>	Input/Output Functions
<stdlib.h>	General Utility Functions
<string.h>	String Functions
<time.h>	Date and Time Functions

Preprocessor Directives

- Constant Macro
  - a name that is replaced by a particular constant value

EX:

```
#define   PI    3.141593
```

constant macro   constant value

```
#define MAX_LENGTH  100
```

Comment

- Two types:
  - One-line comment   //
  - Multiple-line comment /* */

Examples:

```
// This is a one-line comment

/* Hello, this is
   multiple-line comment*/
```

Data Types and Names

- The **C language** reserves some **keywords** words that have **special** meanings to the language.
- Those reserved words should **not** be used as **variables**, **constants**, or **function names** in your program.
- All **C keywords** must be written in **lowercase** letters, for instance **INT** will not be treated as a **keyword**, it must be written as **int**.

The computer list of C keywords

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

Variable declaration

- **Variable**: a name associated with a memory cell whose value can change.

  Examples:

  sum , x ,y , result,…..

Rules for Variables

1. A variable must consist only of letters, digits, and underscores.
2. A variable cannot begin with a digit.
3. A *C reserved word* cannot be used as a user variable.
4. A variable defined in a C standard library should not be redefined.

Reserved Words : A word that has special meaning in C
for example: int, float, double, char , return ,…etc

Variable declarations and data types

Invalid variables names

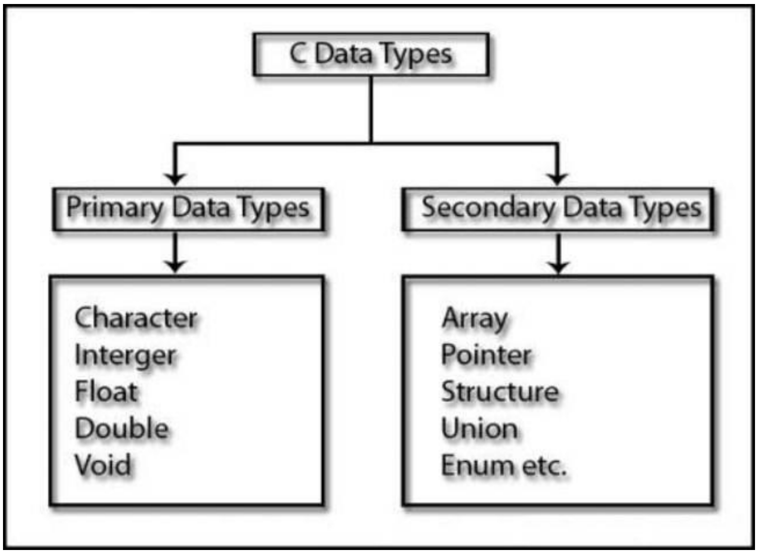
Invalid identifier	Reason Invalid
1Letter	begins with a digit
double	reserved word
int	reserved word
TWO*FOUR	character * not allowed
joe's	character ' not allowed

To remove the ambiguity

Reserved Words	Standard Identifiers	User-Define Identifiers
int	printf	KMS_PER_MILE
void	scanf	miles
double		kms
return		sum

NOTE: Sum, sum, SUM are viewed by the compiler as different identifiers

C Language Data Types



C Data Types

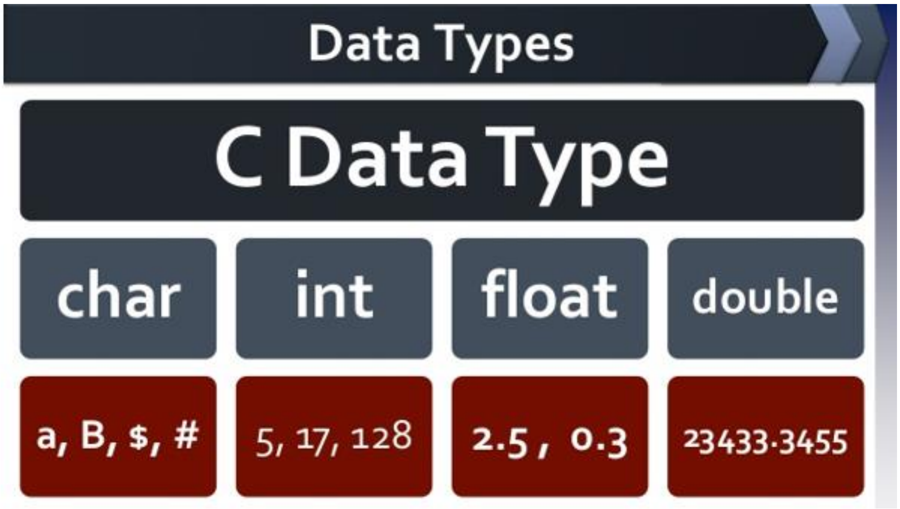
Primary Data Types

Character
Interger
Float
Double
Void

Secondary Data Types

Array
Pointer
Structure
Union
Enum etc.

C Language Data Types - Examples



Data Types

C Data Type

char	int	float	double
a, B, $, #	5, 17, 128	2.5 , 0.3	23433.3455

Variable declarations and data types

Syntax :

– int *variable_list*;

– float *variable_list;*

– double *variable_list;*

– char *variable_list;*

- Examples :

  – int count, large;

  – float ans;   or   float ans=4.2;

  – double x, y, z; or   double x=1.2,y=3.6,z=8.9;

  – char first_initial;

Variables in C

Data types:

- int (16 bit – 2 Bytes)
- float (32 bit – 4 Bytes)
- double (64 bit – 8 Bytes)

- char (8 bit – 1 Byte)
  – represent an individual character value
  – include a letter, a digit, a special symbol
  – ex. ‘A’   ‘z’   ‘2’   ‘9’   ‘ * ’   ‘ : ’ ‘ ” ’   ‘

Real Numbers

- a real number has an integral part and a fractional part that are separated by a decimal point.

- **float** is a 32bit IEEE 754 single precision Floating Point number.
- **double** is a 64bit IEEE 754 double precision floating point number.

Integer Types in C

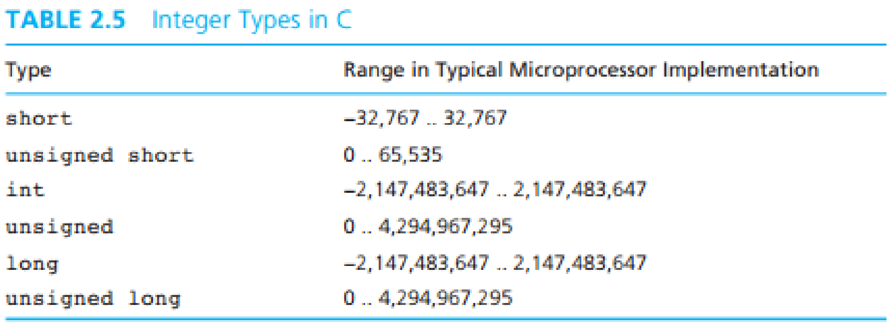
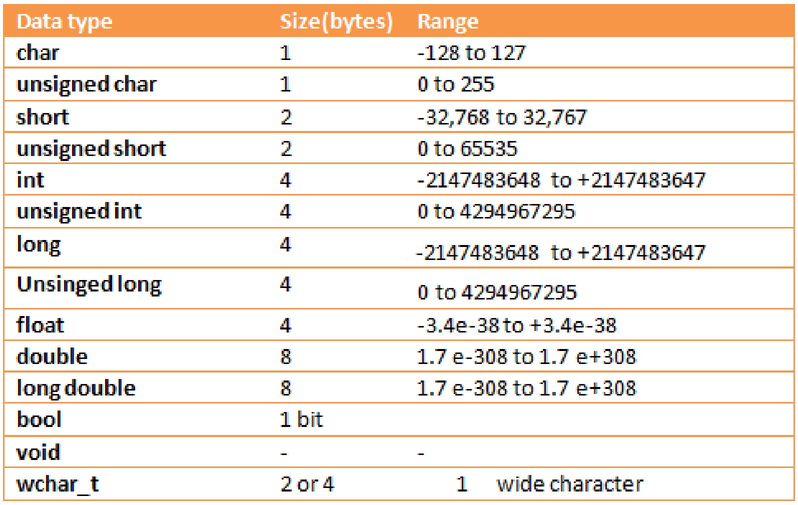


TABLE 2.5 Integer Types in C

Type	Range in Typical Microprocessor Implementation
`short`	−32,767 .. 32,767
`unsigned short`	0 .. 65,535
`int`	−2,147,483,647 .. 2,147,483,647
`unsigned`	0 .. 4,294,967,295
`long`	−2,147,483,647 .. 2,147,483,647
`unsigned long`	0 .. 4,294,967,295

- short – 2 bytes (2^{16} = 65,536)
- int – 4 bytes (2^{32} = 4294967296)

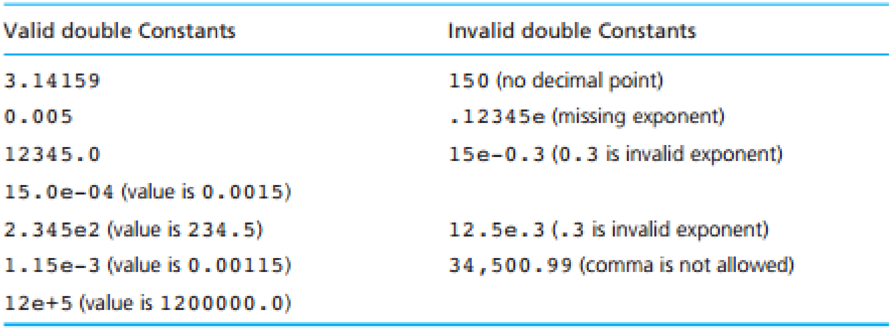
Size of data types in C



Data type	Size(bytes)	Range
char	1	-128 to 127
unsigned char	1	0 to 255
short	2	-32,768 to 32,767
unsigned short	2	0 to 65535
int	4	-2147483648 to +2147483647
unsigned int	4	0 to 4294967295
long	4	-2147483648 to +2147483647
Unsinged long	4	0 to 4294967295
float	4	-3.4e-38 to +3.4e-38
double	8	1.7 e-308 to 1.7 e+308
long double	8	1.7 e-308 to 1.7 e+308
bool	1 bit	
void	-	-
wchar_t	2 or 4	1 wide character

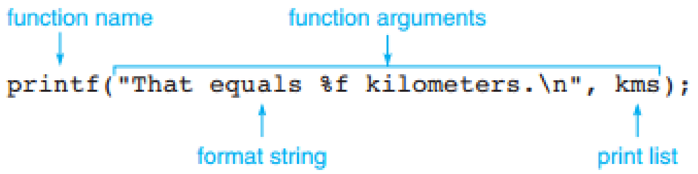
Double Constants

TABLE 2.4 Type double Constants (real numbers)



Valid double Constants	Invalid double Constants
3.14159	150 (no decimal point)
0.005	.12345e (missing exponent)
12345.0	15e-0.3 (0.3 is invalid exponent)
15.0e-04 (value is 0.0015)	
2.345e2 (value is 234.5)	12.5e.3 (.3 is invalid exponent)
1.15e-3 (value is 0.00115)	34,500.99 (comma is not allowed)
12e+5 (value is 1200000.0)	

The printf Function



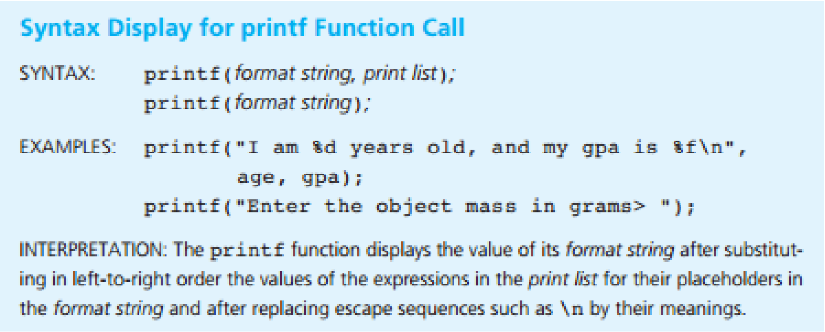
function name

function arguments

```
printf("That equals %f kilometers.\n", kms);
```

format string

print list



Syntax Display for printf Function Call

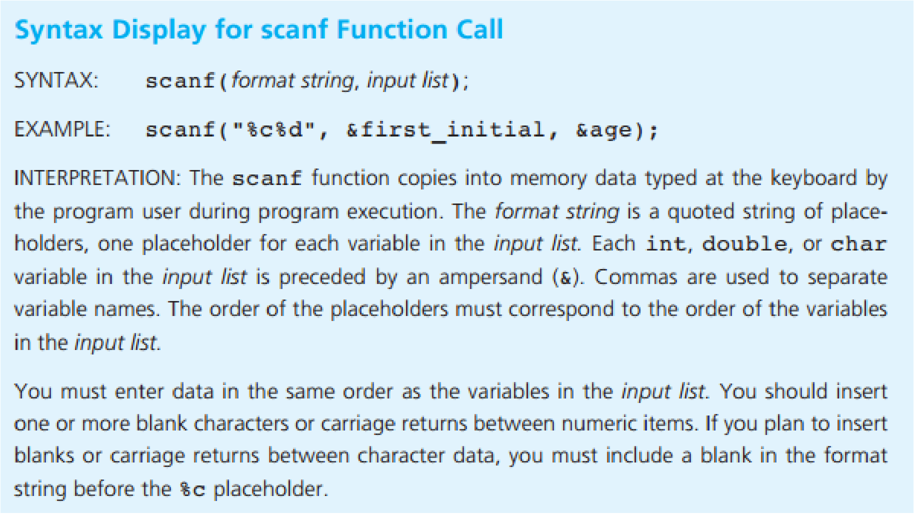
SYNTAX: `printf(`*format string, print list*`);`
`printf(`*format string*`);`

EXAMPLES:

```
printf("I am %d years old, and my gpa is %f\n",
       age, gpa);
printf("Enter the object mass in grams> ");
```

INTERPRETATION: The `printf` function displays the value of its *format string* after substituting in left-to-right order the values of the expressions in the *print list* for their placeholders in the *format string* and after replacing escape sequences such as `\n` by their meanings.

The scanf Function



Syntax Display for scanf Function Call

SYNTAX: `scanf(`*format string, input list*`);`

EXAMPLE: `scanf("%c%d", &first_initial, &age);`

INTERPRETATION: The `scanf` function copies into memory data typed at the keyboard by the program user during program execution. The *format string* is a quoted string of placeholders, one placeholder for each variable in the *input list*. Each `int`, `double`, or `char` variable in the *input list* is preceded by an ampersand (`&`). Commas are used to separate variable names. The order of the placeholders must correspond to the order of the variables in the *input list*.

You must enter data in the same order as the variables in the *input list*. You should insert one or more blank characters or carriage returns between numeric items. If you plan to insert blanks or carriage returns between character data, you must include a blank in the format string before the `%c` placeholder.

Placeholders in Format Strings

Placeholder	Variable Type	Function Use
%c	char	printf / scanf
%d	int	printf / scanf
%f	float	printf / scanf
%f	double	printf
%lf	double	scanf

Placeholders in Format Strings

int : sum
float : a, r
double : num

let sum=2
a=3.2,r=5.2
num= 76.2232

- printf (“The area is %f, a);
- scanf(“ %f ”,&r);
- printf (“the result is %d”, sum);
- scanf (“%lf”,& num);
- printf (“the number is %f”, num)

Arithmetic expressions.

Arithmetic Operator	Meaning	Examples
+	addition	5 + 2 is 7
-	subtraction	5 - 2 is 3
*	multiplication	5 * 2 is 10
/	division	5 / 2 is 2
%	Remainder or Mod	5 % 2 is 1

Arithmetic expressions.

Results of / and % operations

2 / 15 = 0

16 / 3 = 5

4 / 0 undefined

2 % 5 = 2

5 % 4 = 1

15 % 0 undefined

int / int = int

12/3= 4 , 9/8=1

Int/float =float , float/int=float

float/float=float

9/8.0=1.125000

9.0/8=1.125000

9.0/8.0=1.125000

Arithmetic expressions.

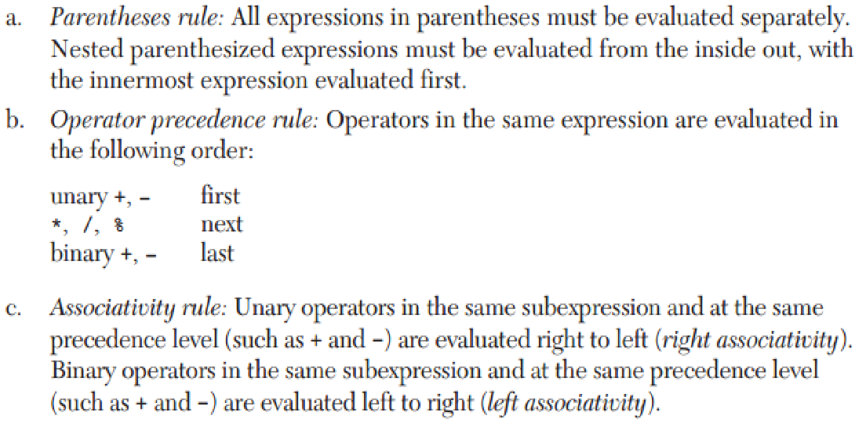
- Example:

```
double k,m;
 k= 9/6;
 m=9/6.0;

 printf("k=%f \nm= %f", k,m);
```

Output:

```
k=1.000000
m=1.500000
```

Arithmetic Expressions – Precedence Rules



a. *Parentheses rule:* All expressions in parentheses must be evaluated separately. Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.

b. *Operator precedence rule:* Operators in the same expression are evaluated in the following order:

unary +, -	first
*, /, %	next
binary +, -	last

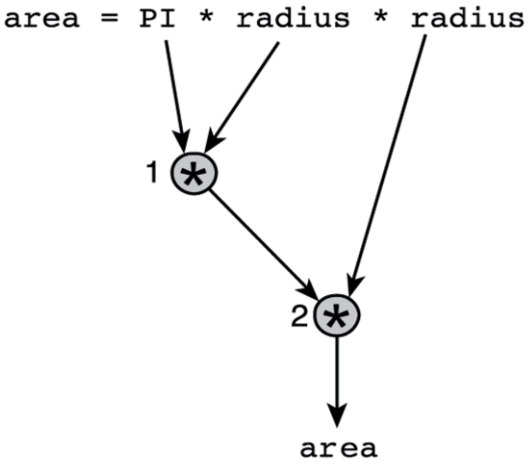
c. *Associativity rule:* Unary operators in the same subexpression and at the same precedence level (such as + and -) are evaluated right to left (*right associativity*). Binary operators in the same subexpression and at the same precedence level (such as + and -) are evaluated left to right (*left associativity*).

Simply:

- ()
- * / %
- + -

Arithmetic expressions.

Example 1 : Evaluate area = PI * radius * radius



Rule c

Arithmetic expressions.

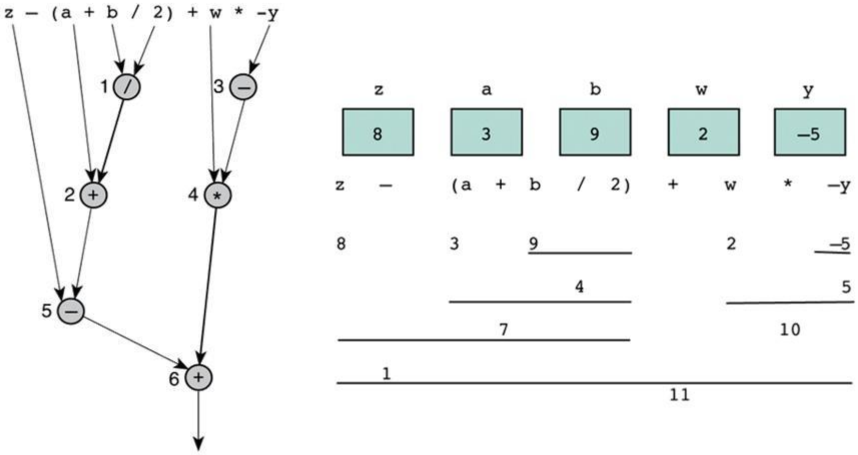
Example 1 : Evaluate area = PI * radius * radius

Let PI= 3.14159 , radius=2.0

```
area     =      PI    *    radius    *    radius
             3.14159          2.0            2.0
             ___________________
                  6.28318
                  _________________________________
                                12.56636
```

Arithmetic expressions.

Example 1 : Evaluate z - (a + b / 2) + w * - y



Arithmetic expressions.

Example:

Write a complete C program that prompts the user to enter the radius of a circle and displays the circumference. Circumference=2 πr

```
#include <stdio.h>
#define PI 3.14159
 int main(void)
  {
      double radius, circum;
      printf("Please enter radius of circle> ");
      scanf("%lf", &radius);
      circum = 2 * PI * radius;
      printf("The circumference is %.2f.\n", circum);
      return  0;
  }
```

Mathematical Formula as C Expression

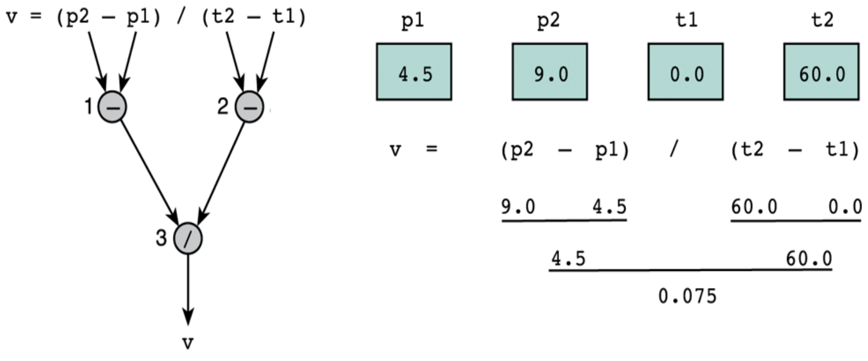
Mathematical Formula	C Expression
b^2-4ac	b * b - 4 * a * c
a + b - c	a + b - c
$\frac{a+b}{c+d}$	(a + b) / (c + d)
$\frac{1}{1+x^2}$	1 / (1 + x * x)
a x -(b + c)	a * -(b + c)

- Always specify multiplication explicitly by using the operator * where needed (formulas 1 and 4).
- Use parentheses when required to control the order of operator evaluation (formulas 3 and 4).
- Two arithmetic operators can be written in succession if the second is a unary operator (formula 5).

Mathematical Formula - Example

Example 1 : Evaluate $v = \frac{p2-p1}{t2-t1}$

let P1=4.5 ,P2=9.0, t1=0.0, t2=60.0



Formatting Values of Type int

int x= 4678, y=3 , z=19

1. printf (“%d   %d %d”, x,y,z)

```
Output

4678▓▓3▓▓ 19
```

2. printf (“%7d %5d %6d”, x,y,z)

```
Output
▓ ▓ ▓ 4678▓▓▓ ▓ ▓ ▓ 3▓ ▓ ▓ ▓ ▓ ▓  19
```

Formatting Output (Practice)

TABLE 2.14 Displaying 234 and –234 Using Different Placeholders

Value	Format	Displayed Output		Value	Format	Displayed Output
234	%4d	▓234		-234	%4d	-234
234	%5d	▓▓234		-234	%5d	▓-234
234	%6d	▓▓▓234		-234	%6d	▓▓-234
234	%1d	234		-234	%2d	-234

Formatting Values of Type **float**

- float x=56.2757   y=2.3849 z=114.2
  printf (“%8.3f%-7.2f%7.4f”,x,y,z);

```
▨ ▨ 56.276 2.38▨ ▨ ▨114.2000
```

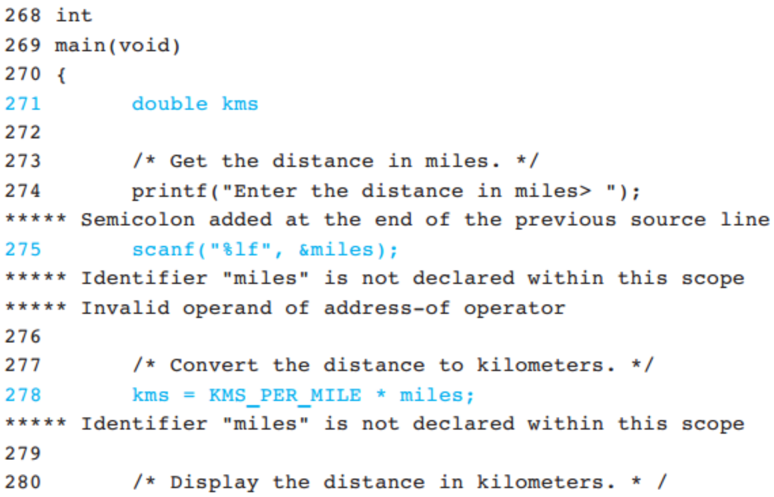
Formatting Output (Practice)

Value	Format	Displayed Output	Value	Format	Displayed Output
3.14159	%5.2f	▨3.14	3.14159	%4.2f	3.14
3.14159	%3.2f	3.14	3.14159	%5.1f	▨ ▨ 3.1
3.14159	%5.3f	3.142	3.14159	%8.5f	▨3.14159
.1234	%4.2f	0.12	-.006	%4.2f	-0.01
-.006	%8.3f	▨▨-0.006	-.006	%8.5f	-0.00600
-.006	%.3f	-0.006	-3.14159	%.4f	-3.1416

Error Types in C

1. **<u>Syntax Error:</u>** a violation of the C grammar rules, detected during program compilation.

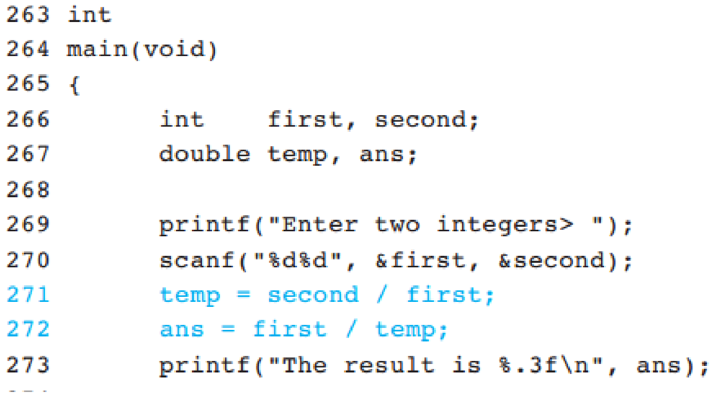
<u>Example</u>:



```
int
main(void)
{
          double kms

          /* Get the distance in miles. */
          printf("Enter the distance in miles> ");
***** Semicolon added at the end of the previous source line
          scanf("%lf", &miles);
***** Identifier "miles" is not declared within this scope
***** Invalid operand of address-of operator

          /* Convert the distance to kilometers. */
          kms = KMS_PER_MILE * miles;
***** Identifier "miles" is not declared within this scope

          /* Display the distance in kilometers. * /
```

2. **<u>Run-time errors</u>**: detected and displayed by the computer during the execution of a program

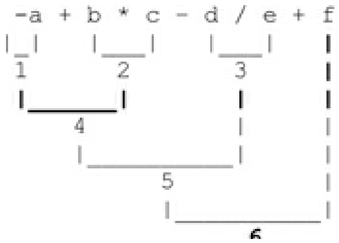
A run-time error occurs when the program directs the computer to perform <u>an illegal operation</u>, such as *dividing a number by zero*

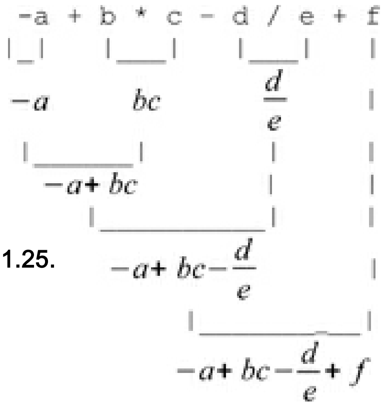


```
int
main(void)
{
          int    first, second;
          double temp, ans;

          printf("Enter two integers> ");
          scanf("%d%d", &first, &second);
          temp = second / first;
          ans = first / temp;
          printf("The result is %.3f\n", ans);
```

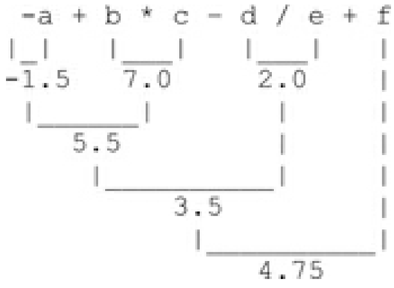
3. Logic Errors: An error caused by following an incorrect algorithm.

LAB: Evaluation of Simple Arithmetic Expressions

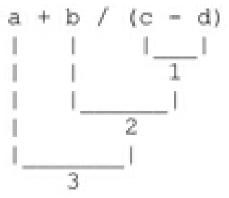




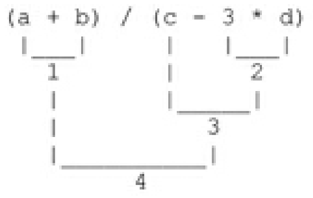
a = 1.5, b = 2.0, c = 3.5, d = 5.0, e = 2.5 and f = 1.25.



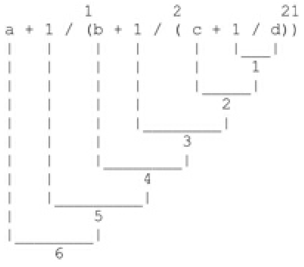
LAB: Evaluation of simple parenthesized expressions



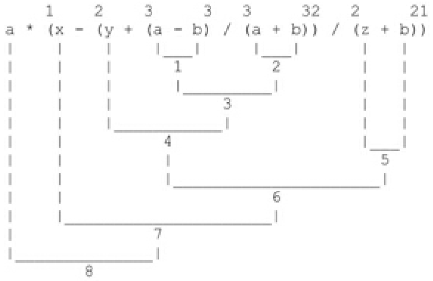
(a)



(b)



(a)



(b)

Extra Exercises

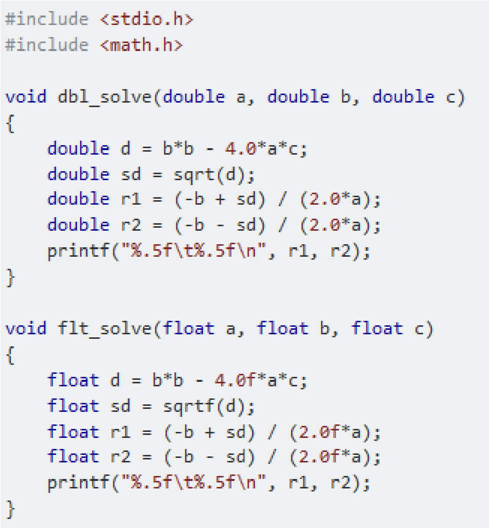
1. Which of the following identifiers are (a) C reserved words, (b) standard identifiers, (c) conventionally used as constant macro names, (d) other valid identifiers, and (e) invalid identifiers?

void	MAX_ENTRIES	double	time	G	Sue's
return	printf	xyz123	part#2	"char"	#insert
this_is_a_long_one					

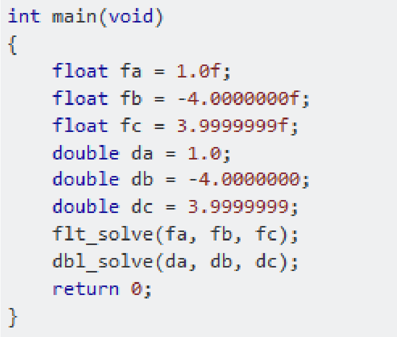
2. Do a step-by-step evaluation of the expressions that follow if the value of celsius is 38.1 and salary is 38450.00 .

- 1.8 * Celsius + 32.0
- (salary - 5000.00) * 0.20 + 1425.00

Given a quadratic equation: $X^2 - 4.0000000\ x + 3.9999999 = 0$. Using float and double, we can write a test program:

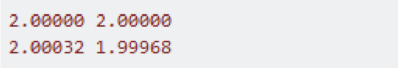


```
#include <stdio.h>
#include <math.h>

void dbl_solve(double a, double b, double c)
{
    double d = b*b - 4.0*a*c;
    double sd = sqrt(d);
    double r1 = (-b + sd) / (2.0*a);
    double r2 = (-b - sd) / (2.0*a);
    printf("%.5f\t%.5f\n", r1, r2);
}

void flt_solve(float a, float b, float c)
{
    float d = b*b - 4.0f*a*c;
    float sd = sqrtf(d);
    float r1 = (-b + sd) / (2.0f*a);
    float r2 = (-b - sd) / (2.0f*a);
    printf("%.5f\t%.5f\n", r1, r2);
}
```



```
int main(void)
{
    float fa = 1.0f;
    float fb = -4.0000000f;
    float fc = 3.9999999f;
    double da = 1.0;
    double db = -4.0000000;
    double dc = 3.9999999;
    flt_solve(fa, fb, fc);
    dbl_solve(da, db, dc);
    return 0;
}
```

Running the program gives me:



```
2.00000 2.00000
2.00032 1.99968
```

The exact roots to 10 significant digits are, r1 = 2.000316228 and r2 = 1.999683772.