





# Programming with C I





Fangtian Zhong  
CSCI 112

Gianforte School of Computing  
Norm Asbjornson College of Engineering  
E-mail: [fangtian.zhong@montana.edu](mailto:fangtian.zhong@montana.edu)



# Objectives

-  To become familiar with the general form of a C program and the basic elements in a program.
-  To appreciate the importance of writing comments in a program.
-  To understand the use of data types and the differences between the data types int, double, and char.
-  To know how to declare variables.

# Objectives

-  To understand how to write assignment statements to change the value of variables.
-  To learn how C evaluates arithmetic expressions and how to write them in C.
-  To learn how to read data values into a program and to display results.
-  To understand how to write format strings for data entry and display.

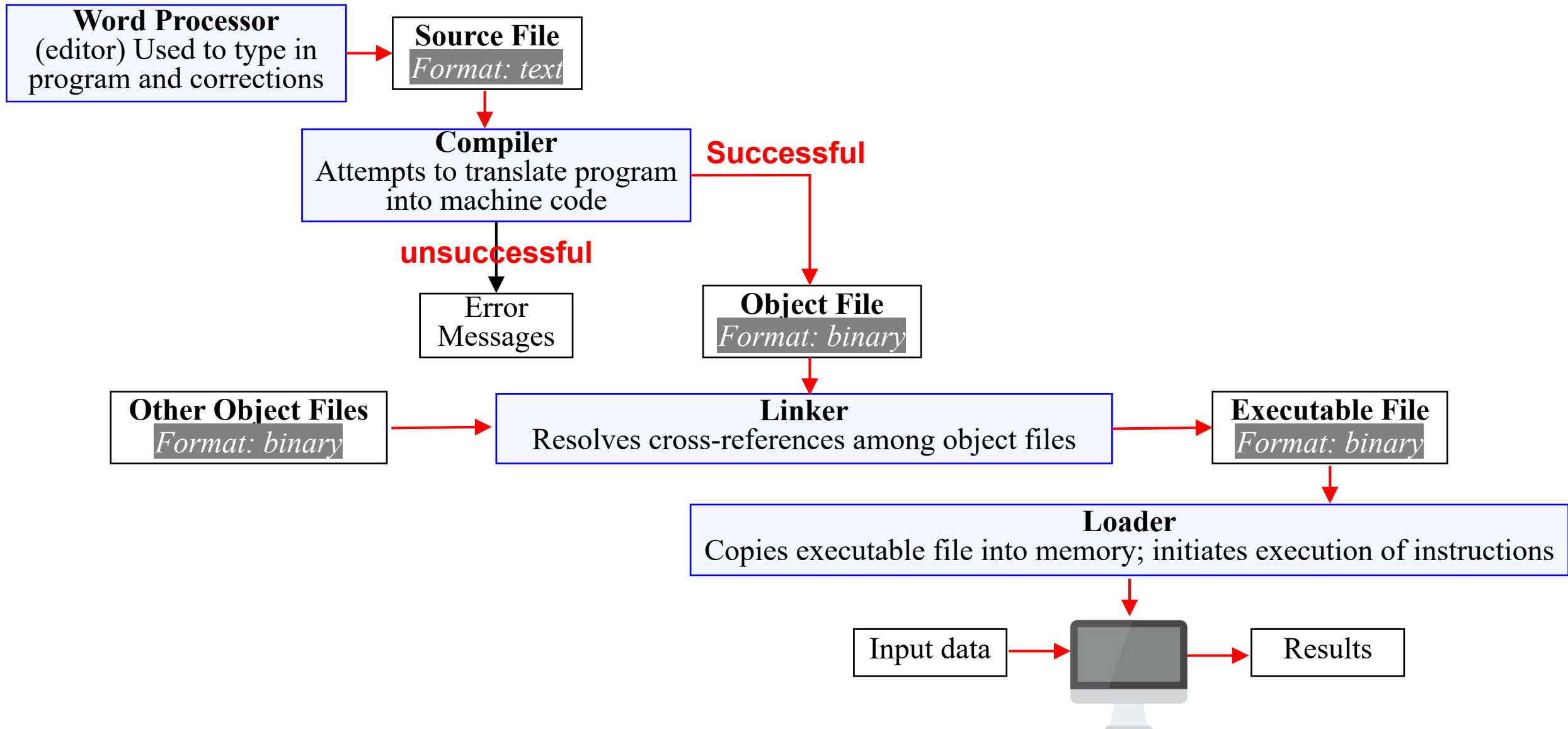
# Objectives

-  To learn how to use redirection to enable the use of files for input/output.
-  To understand the differences between syntax errors, run-time errors, and logic errors, and how to avoid them and to correct them.

# C

- A high-level programming language.
- Developed in 1972 by Dennis Ritchie at AT&T Bell Labs.
- Designed as the language to write the Unix operating system.
- Resembles everyday English.
- Very popular.

# Entering, Translating and Running a High-Level Language Program



# Language Elements

## **preprocessor**

- a system program that modifies a C program prior to its compilation.

## **library**

- a collection of useful functions and symbols that may be accessed by a program.
- each library has a standard header file whose name ends with the symbols “.h”.

**stdio.h**

# Language Elements

## **preprocessor directive**

- a C program line beginning with # that provides an instruction to the preprocessor.

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

```
#define KMS_PER_MILE 1.609
```



# Language Elements

## **constant macro**

- a name that is replaced by a particular constant
- value before the program is sent to the compiler

```
#define KMS_PER_MILE 1.609
```

*constant*



*constant macro*



```
kms = KMS_PER_MILE * miles;
```

# Language Elements

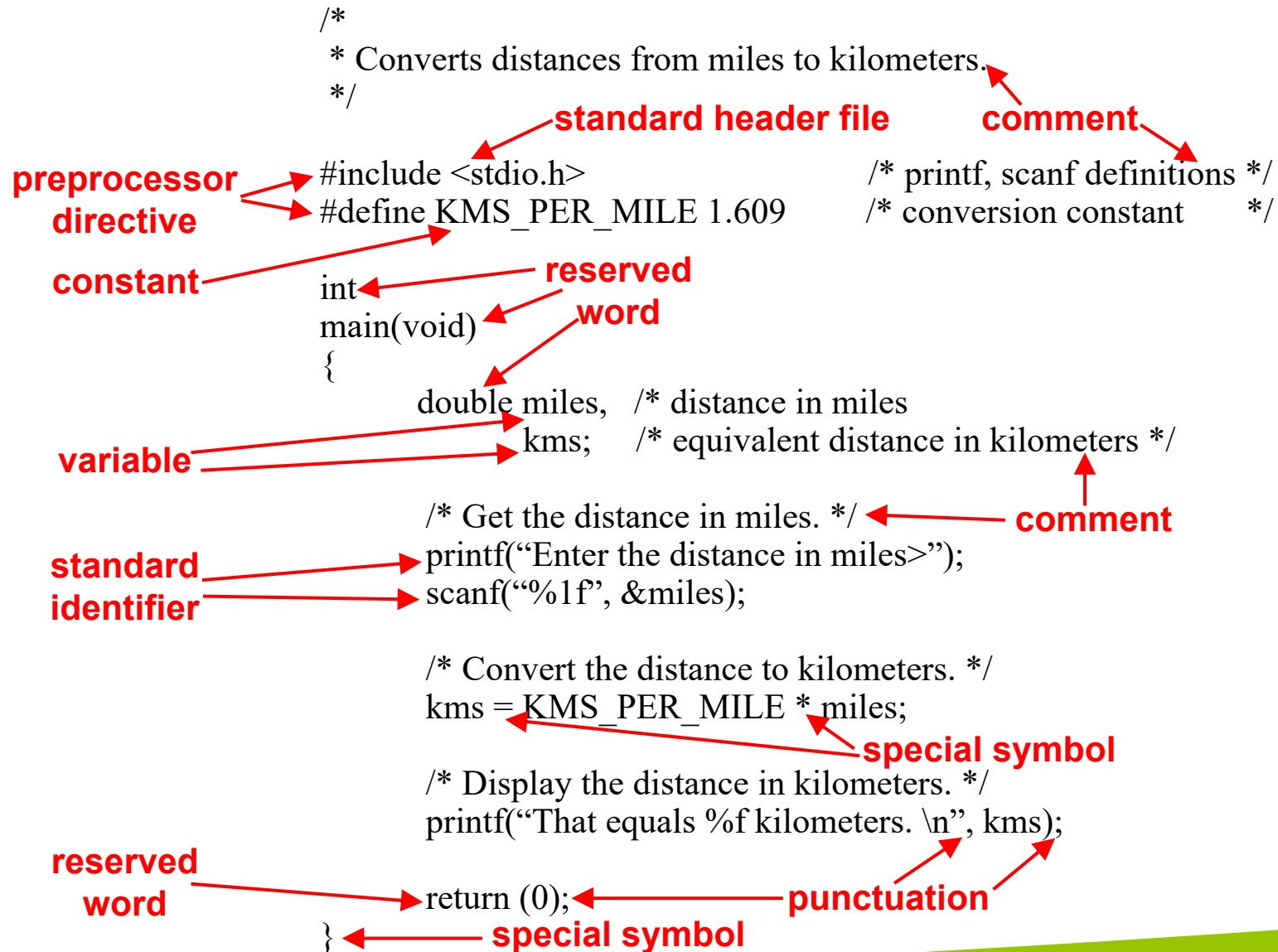


## comment

- text beginning with `/*` and ending with `*/` that provides supplementary information but is ignored by the preprocessor and compiler.
- for single-line comments, can use `//`

```
/* Get the distance in miles */  
// Get the distance in miles
```


# C Language Elements in Miles-to-Kilometers Conversion Program



# Function main

 Every C program has a main function.

```
int main (void)
```

 These lines mark the beginning of the main function where program execution begins.

# Function main



## **declaration**

- the part of a program that tells the compiler the names of memory cells in a program.



## **executable statements**

- program lines that are converted to machine language instructions and executed by the computer.

# Variable Declarations

## **variable**

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

## **variable declarations**

- statements that communicate to the compiler the names of variables in the program and the kind of information stored in each variable.

# Variable Declarations

- 🛡️ C requires you to declare every variable used in a program.
- 🛡️ A variable declaration begins with an identifier that tells the C compiler the type of data store in a particular variable.

```
int hours;
```

```
double miles;
```

# Data Types

## **int**

- a whole number
- 435

## **double**





- a real number with an integral part and a fractional part separated by a decimal point
- 3.14159

## **char**

- an individual character value
- enclosed in single quotes
- 'A', 'z', '2', '9', '\*', '!'



# Executable Statements

-  Follow the declarations in a function.
-  Used to write or code the algorithm and its refinements.
-  Are translated into machine language by the compiler.
-  The computer executes the machine language version.

# Executable Statements



## assignment statemen

- an instruction that stores a value of a computational result in a variable

```
kms = KMS_PER_MILE * miles;
```

# Executable Statements

- Assignment is not the same as an algebraic equation.
- The expression to the right of the assignment operator is first evaluated.
- Then the variable on the left side of the assignment operator is assigned the value of that expression.

```
sum = sum + item;
```

# The **printf** Function

- 🛡️ Displays a line of program output.
- 🛡️ Useful for seeing the results of a program execution.

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

# The **printf** Function



## function argument

- enclosed in parentheses following the function name
- provides information needed by the function

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

**function name**

# The **printf** Function



## format string

- in a call to **printf**, a string of characters enclosed in quotes, which specifies the form of the output line

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

# The **printf** Function

## **print list**

- in a call to **printf**, the variables or expressions whose values are displayed

## **placeholder**

- a symbol beginning with % in a format string that indicates where to display the output value

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

# Formatting Numbers in Program Output



## field width

- the number of columns used to display a value



## No. of decimal places



When formatting doubles, you may indicate the total field width needed and the number of decimal places desired.

```
printf("Your result equals %5.1f kilometers. \n", kms);
```



# Let's write a C program

**That stores an int, double, and char variable, and prints them all out.**




# Placeholders in format string

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

# The **scanf** Function

-  Copies data from the standard input device (usually the keyboard) into a variable.

```
scanf( "%lf" , &miles);  
scanf( "%c%c%c" , &letter_1, &letter_2, &letter_3);
```

-  Must pass address of variable to store using the addressof operator (&)

# The **return** Statement

- 🛡️ Last line in the main function.
- 🛡️ Transfers control from your program to the operating system.
- 🛡️ The value 0 indicates that your program executed without an error.

```
return (0);
```

# Arithmetic Operators

Arithmetic Operator	Meaning	Example
+	addition	$5 + 2$ is 7 $5.0 + 2.0$ is 7.0
-	subtraction	$5 - 2$ is 3 $5.0 - 2.0$ is 3.0
*	multiplication	$5 * 2$ is 10 $5.0 * 2.0$ is 10.0
/	division	$5.0 / 2.0$ is 2.5 $5 / 2$ is 2
%	remainder	$5 \% 2$ is 1

# Type casting



converting an expression to a different type by writing the desired type in parentheses in front of the expression

```
int x = 5;  
double y = (double) x;
```

# Rules for Evaluating Expressions



## Parentheses rule

- all expression must be evaluated separately
- nested parentheses evaluated from the inside out
- innermost expression evaluated first



## Operator precedence rule

- unary +, - first (setting sign)
- \*, /, % next
- binary +, - last



## Note prefix and postfix increment/decrement!

- ++a and --a are executed before value is used
- a++ and a-- are executed after value is used

# Rules for Evaluating Expressions



## Right Associativity

- Unary operators in the same subexpression and at the same precedence level are evaluated right to left.



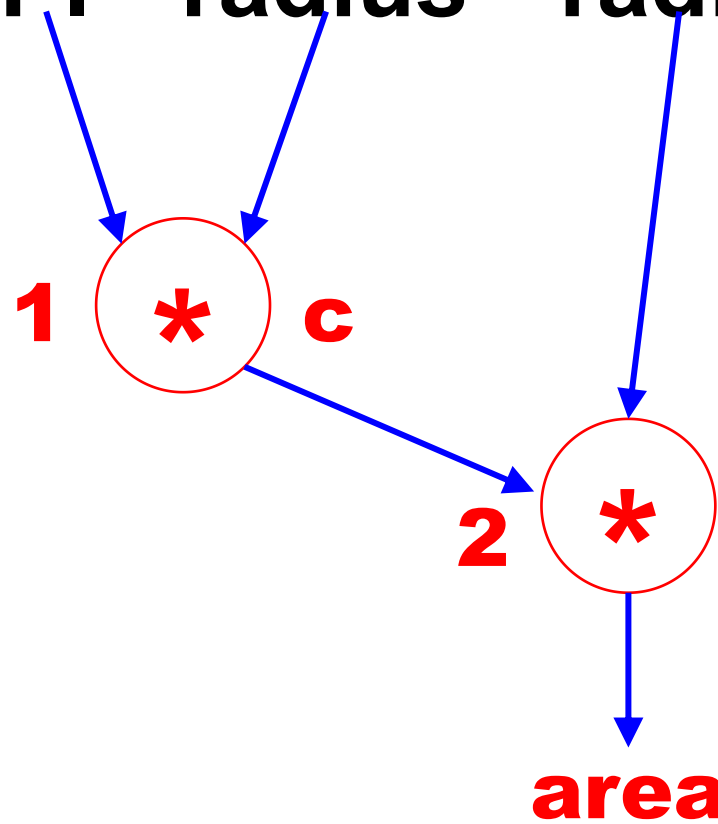
## Left Associativity

- Binary operators in the same subexpression and at the same precedence level are evaluated left to right.



# Figure Evaluation Tree for $\text{area} = \text{PI} * \text{radius} * \text{radius};$

$\text{area} = \text{PI} * \text{radius} * \text{radius}$

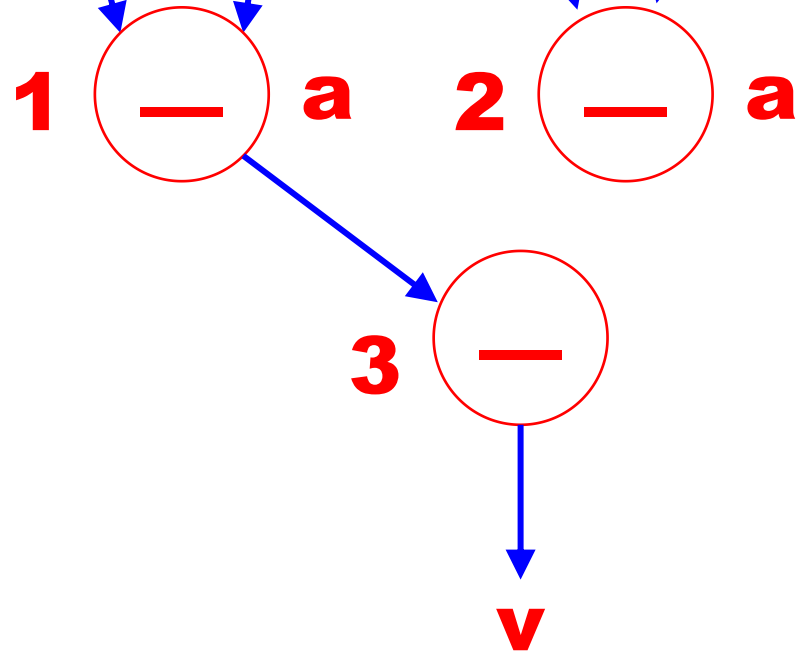


# Figure Step-by-Step Expression Evaluation

$$\begin{array}{r} \text{area} = \text{PI} * \text{radius} * \text{radius} \\ \quad \quad \quad \underline{3.14159} \quad \quad 2.0 \quad \quad 2.0 \\ \quad \quad \quad \quad \quad \quad \underline{6.28318} \\ \quad \quad \quad \quad \quad \quad \quad \quad \quad 12.56636 \end{array}$$

**Figure** Evaluation Tree and Evaluation for  $v = (p2 - p1) / (t2 - t1)$ ;

$$v = (p2 - p1) / (t2 - t1)$$

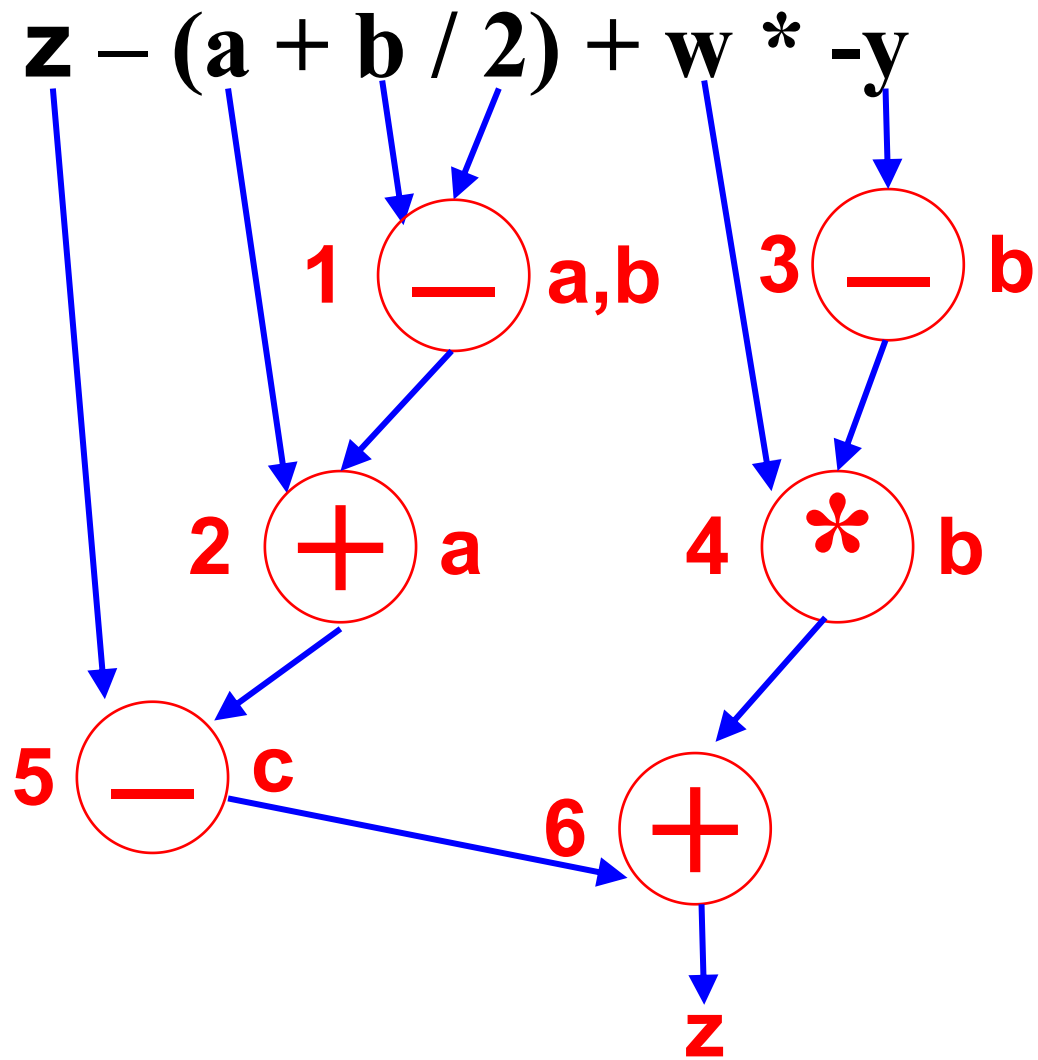


p1	p2	t1	t2
4.5	9.0	0.0	60.0

$$v = (p2 - p1) / (t2 - t1)$$

<u>9.0</u>	<u>4.5</u>	<u>60.0</u>	<u>0.0</u>
<u>4.5</u>		<u>60.0</u>	
0.075			

# Figure Evaluation Tree and Evaluation for $z - (a + b / 2) + w * -y$



z	a	b	w	y
8	3	9	2	-5
z	(a + b / 2)	+ w * -y		
8	3	9	2	-5
		4		5
		7		10
1				
		11		

# Common Programming Errors



## debugging

- removing errors from a program



## syntax error

- a violation of the C grammar rules
- detected during program translation (compilation)



## run-time error

- an attempt to perform an invalid operation
- detected during program execution



## logic error

- an error caused by following an incorrect algorithm

# Figure A Program with a Run-Time Error

```
#include <stdio.h>

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

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

    return (0);
}
```

Enter two integers> 14 3

Arithmetic fault, divide by zero at line 272 of routine main

## Figure A Program That Produces Incorrect Results Due to & Omission

```
#include <stdio.h>




int
main (void)
{
    int  first, second; sum;

    printf("Enter two integers> ");
    scanf("%d%d", first, second); /* ERROR || should be &first, &second */
    sum = first + second;
    printf("%d + %d = %d\n", first, second, sum);

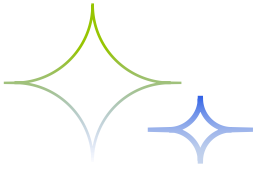
    return (0);
}
```

```
Enter two integers> 14 3
5971289 + 5971297 = 11942586
```

# Wrap Up

-  Every C program has preprocessor directives and a main function.
-  The main function contains variable declarations and executable statements.
-  C's data types enable the compiler to determine how to store a value in memory and what operations can be performed on that value.





# THE END

Fangtian Zhong  
CSCI 112

Gianforte School of Computing  
Norm Asbjornson College of Engineering  
E-mail: [fangtian.zhong@montana.edu](mailto:fangtian.zhong@montana.edu)