Computer Programming

Introduction to
C Programming Language

History of C

- C Programming Language
  - Evolved by Ritchie from two previous programming languages, BCPL and B
  - Used to develop UNIX
  - Used to write modern operating systems
  - Hardware independent (portable)
  - By late 1970's C had evolved to "Traditional C“
- Standardization
  - Many slight variations of C existed, and were incompatible
  - Committee formed to create a "unambiguous, machine-independent" definition
  - Standard created in 1989, updated in 1999

Portability Tips

- Because C is a hardware-independent, widely available language, applications written in C can run with little or no modifications on a wide range of different computer systems.

C Standard Library

- C programs consist of pieces/modules called functions
  - A programmer can create his own functions
    - Advantage: the programmer knows exactly how it works
    - Disadvantage: time consuming
  - Programmers will often use the C library functions
    - C defines a small number of operations, instead it contains usefull libraries
    - Use these as building blocks
  - Avoid re-inventing the wheel
    - If a pre-made function exists, generally best to use it rather than write your own
    - Library functions carefully written, efficient, and portable

Performance Tips

- Using Standard C library functions instead of writing your own comparable versions can improve program performance, because these functions are carefully written to perform efficiently.

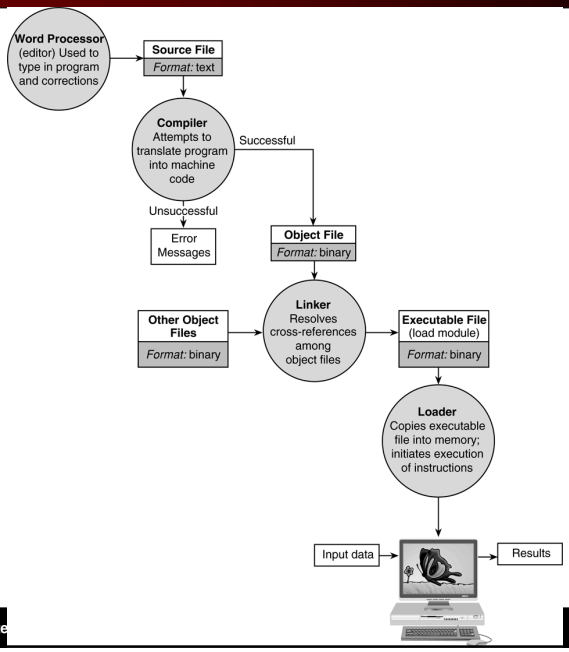
Portability Tips

- Using Standard C library functions instead of writing your own comparable versions can improve program portability, because these functions are used in virtually all Standard C implementations.

Software Engineering Observation

- Read the manuals for the version of C you are using. Reference these manuals frequently to be sure you are aware of the rich collection of C features and that you are using these features correctly.

Typical C Program Development



Software Engineering Observation

- Your computer and compiler are good teachers. If you are not sure how a feature of C works, write a sample program with that feature, compile and run the program and see what happens.

Software Development

- Programming = problem solving
- Methodology
  - Specify the problem requirements
  - Analyze the problem
  - Design an algorithm
  - Implement the algorithm
  - Test and verify the program
  - Maintain and update the program

Problem Requirements

- Statements of the problem
  - Understand the problem
  - Retrieve the requirements
  - Eliminate unimportant aspects
- May need to get information from specialists
- EX: Write a program for mile to kilometer conversion

Analysis

- Identify
  - Input data
  - Output data
  - Additional requirements and constraints
- Decide aspects of data
  - Representation
  - Relationships
- EX:
  - Input: distance on miles
  - Output: distance on kilometers
  - Representation: floating point numbers
  - Relationship: 1 mile = 1.609 kilometers

Designing algorithm

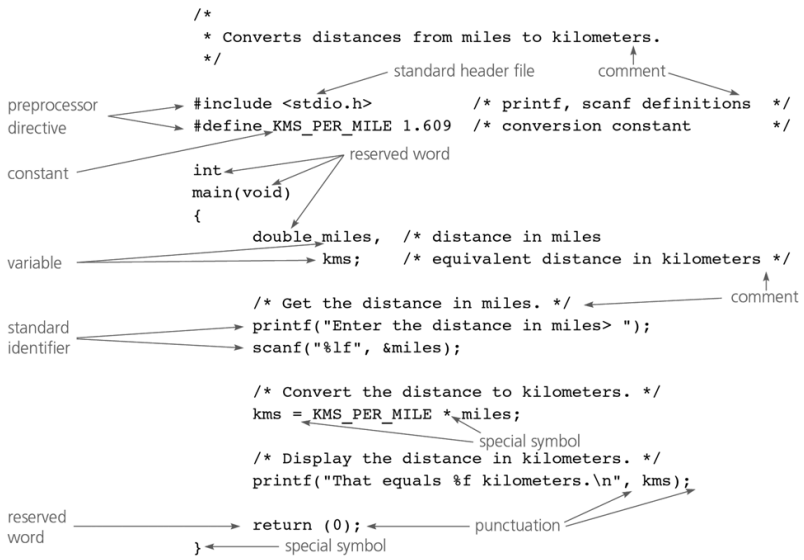
- Top-down stepwise refinement
  - List major steps (sub-problems)
  - Break down each step into a more detailed list
- Desk-check your algorithm
  - Perform steps of the algorithm by yourself
- Ex:
  1. Get the distance in miles
  2. Convert the distance to kilometers
  3. Display the distance in kilometers
  - Step 2 may require further refinement

    2.1 The distance in kilometers is 1.609 times the distance in miles

Software Development

- Methodology
  - Specify the problem requirements
  - Analyze the problem
  - Design an algorithm
  - Implement the algorithm
    - Writing the algorithm in C by converting each step into statements of C
  - Test and verify the program
    - Run the program for several input cases
  - Maintain and update the program
    - Keep the program up-to-date

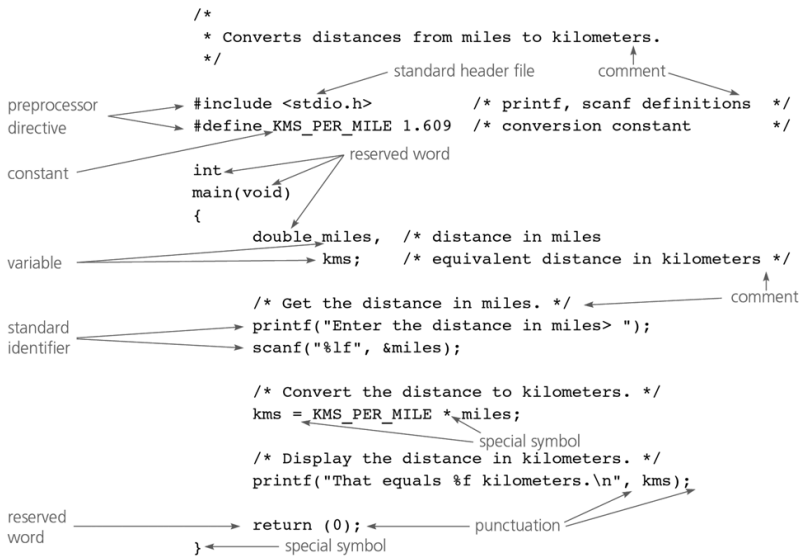
C Language Elements



Preprocessor Directives

- Preprocessor modifies the text of a C program before compilation
- Preprocessor directives
  - Start with a #
- #include <stdio.h>
  - Each library has a header file. Include it to access the library
  - Preprocessor inserts definitions from the header
  - stdio.h includes information about standard input/output
- #define KMS_PER_MILE 1.609
  - Defines a constant macro
    - Value of KMS_PER_MILE can not change
  - Preprocessor replaces each occurrence of “KMS_PER_MILE” in the text with “1.609”
  - KMS_PER_MILE is easier to remember

C Language Elements



Function main

- C programs have exactly one main function
  - Marks the beginning of program execution
  - (void) indicates that function receives no data
  - int means that main "returns" an integer value
- Function bodies enclosed in braces ({ and })
  - Function body has two parts
    - Declaration
    - Executable statements

Identifiers

- Reserved words
  - Ex: “int” and “void”
  - Can not be used for any other purpose
- Standard identifiers
  - Ex: scanf, printf
  - Has a special meaning but can be redefined
- User-defined identifiers
  - Ex: name of memory cells (miles) and KMS_PER_MILE
  - Free to select the name
  - Syntax rules:
    - Includes only letters, digits and underscore
    - Can not begin with digit
- C is case sensitive!

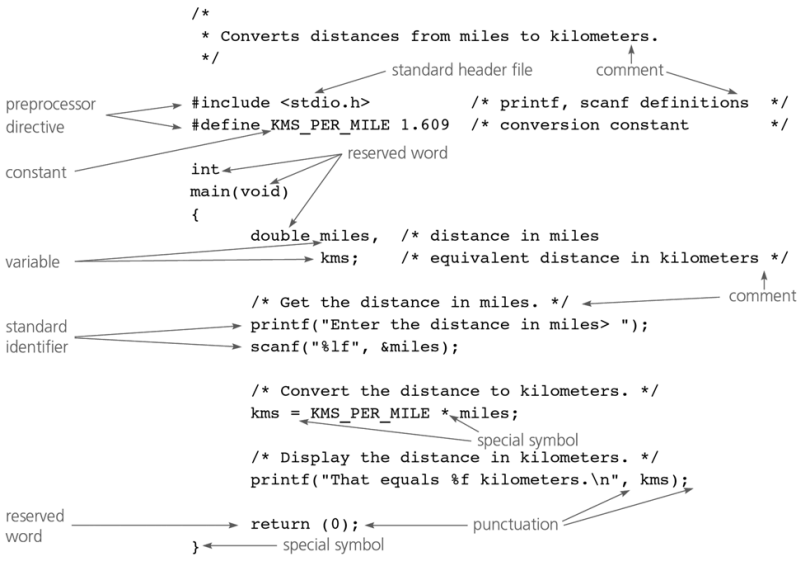
Reserved words

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

Program Style

- Pick meaningful identifiers
  - Long enough to convey the meaning
- If the identifier consists of two words, place an underscore character between words
- Do not choose similar identifier names
- Use uppercase letters for names of macros
  - Use lowercase letters otherwise

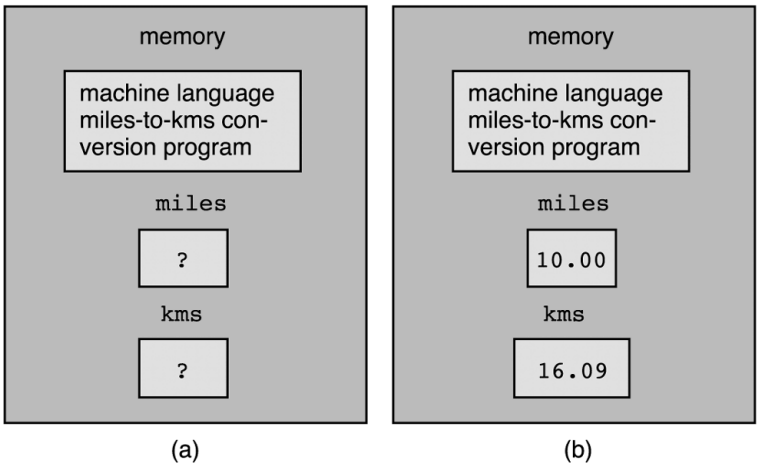
C Language Elements



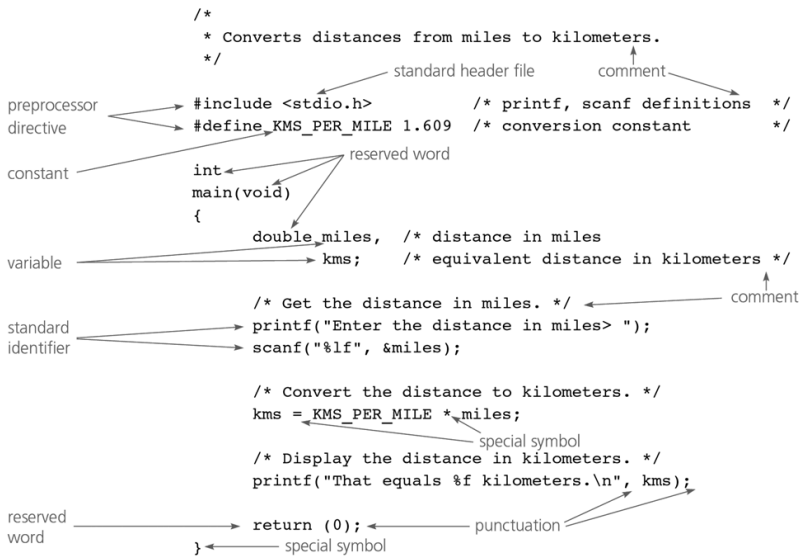
Variables

- Variables: memory cells for storing data
  - Values can change
- Every variable has:
  - Name: identifier
  - Type: int, double, char
  - Size
  - Value
- Data types: abstraction of real types
  - Predefined data types
  - User-defined data types
  - Each constant or variable has a type
  - int: whole numbers (-123, 15, 27384)
    - There is a range because of finite memory cell size
  - double: real numbers (12.345, 0.5217e-4)
    - Too large and too small numbers can not be represented
  - char: character values (‘a’, ‘5’, ‘^’, ‘,’)

Memory (a) Before and (b) After Execution



C Language Elements



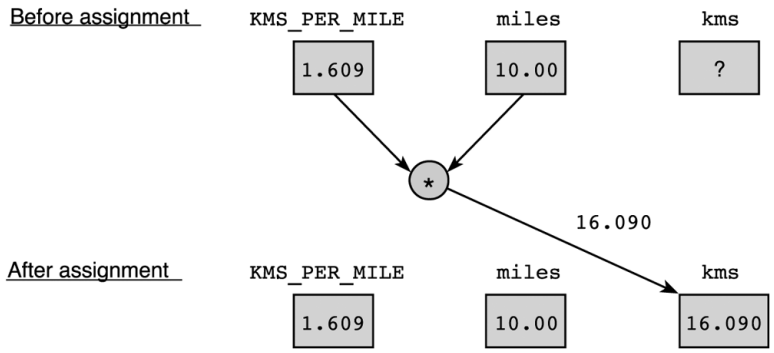
Executable Statements

- Comes after declaration
- Compiler translates to machine language code

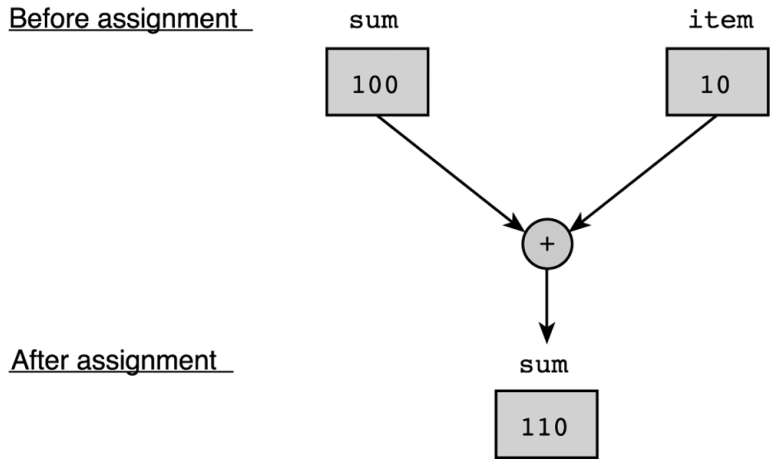
- Assignment statements
  - Used to store value to a variable
  - Ex: kms = KMS_PER_MILE * miles;
  - In general

    variable = expression;
  - Assignment operator: =
    - Should be read as
      - becomes
      - gets
      - takes a value of
  - Previous value of variable is lost!..

Effect of kms = KMS_PER_MILE * miles;



Effect of sum = sum + item;



Executable Statements

- Input/Output Operations
  - Input Operation: Reading a value into a variable by scanf
    - A different data can be entered by the user
  - Output Operation: Displaying a value by printf
- Several I/O functions in C
  - All in standard I/O library

    #include <stdio.h>
- Function call is an executable statement.
  - Function performs the action for you.

printf

Displays the output

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

printf(format string, print list);

- Function name: printf
- Function arguments: in paranthesis
  - Format string: "That equals %f kilometers.\n"
  - Print list: kms
- Placeholders: %c, %d, %f, %lf
- Escape sequence:
  - \n means newline : cursor moves the beginning of the next line
  - Can be used anywhere in the format string

Escape sequences

Escape sequence	Description
\n	Newline. Position the cursor at the beginning of the next line.
\t	Horizontal tab. Move the cursor to the next tab stop.
\a	Alert. Sound the system bell.
\\	Backslash. Insert a backslash character in a string.
\"	Double quote. Insert a double-quote character in a string.

scanf

Reads the data into a variable

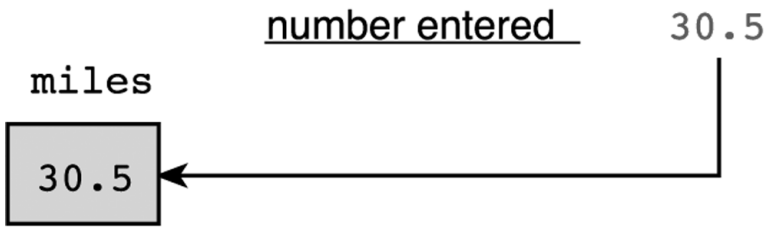
scanf("%lf", &miles);

scanf(format string, input list);

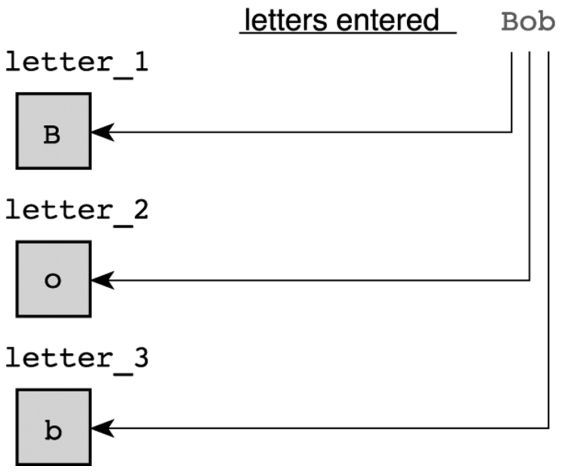
- Function name: scanf
- Function arguments: in paranthesis
  - Format string: "%lf"
  - Input list: &miles

- Address-of operator: &
  - Used to inform scanf about the location of the variable
  - In not used scanf knows only the value of the variable

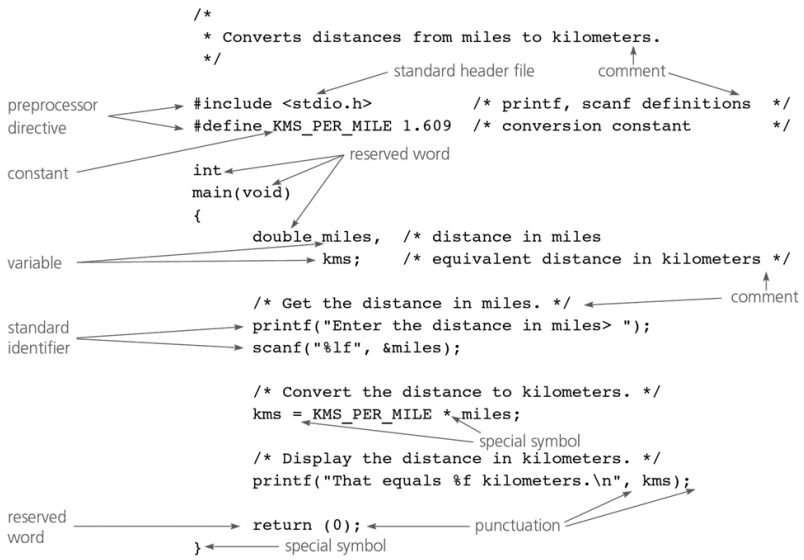
Effect of scanf("%lf", &miles);



Scanning Data Line Bob



C Language Elements



Others

- The return statement
  - Transfers the control to the OS
  - Return value indicates whether the operation is successful or not
- Comments
  - Ignored by the compiler
  - Provides information for the programmer

```
/*  this is a comment */
```

General Form of a C Program

preprocessor directives
main function heading
{
    declarations
    executable statements
}

Program Style

- One statements in each line
- Use extra spaces for readability
  - Compiler ignores them
  - Leave a space before and after operators
  - Indent each block
  - Insert blank lines between sections
- Use comments
  - Write a descriptive comment for
    - the program (header section)
    - each identifier
    - each program section

Arithmetic Expressions

- Manipulates type int and double data
- Binary operators: +, -, *, /, %
  - Two operand: constant, variable or expression
  - Type of the result depend on the types of the operands
    - int if both operands are integer
    - double otherwise
    - Mixed type expression???
- / operator
  - Integer division: computes integral part of the division
  - Division by zero!..
- % operator
  - Returns integer remainder
  - Zero right operand? Undefined!..
  - Negative right operand is non standard

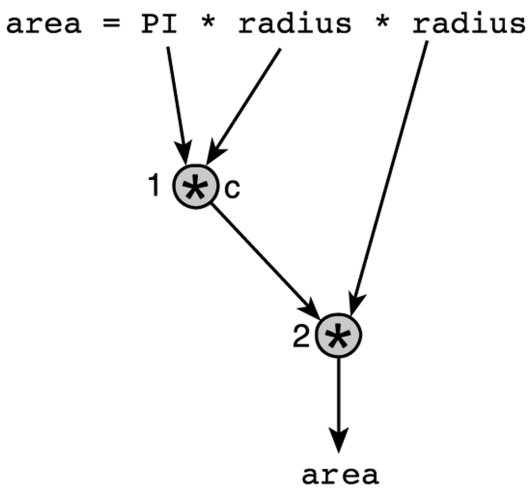
Arithmetic Expressions

- Unary Operators: +, -
  - One operand
- Assignment:
  - The value of expression is evaluated and result is assigned
  - What if the type of the expression and the type of the variable is different?
    - Assignment of int to double
      - Fractional part id zero
    - Assignment of double to int
      - Fractional part is lost
    - Automatic type conversion
    - Type casting
      (int) 3.7

Expression Evaluation

- If there are multiple operators in an expression the order of evaluation makes a difference
  - Ex: x / y * z
- Evaluation Rules:
  - Parenthesis rule:
    - All expressions in parenthesis are evaluated seperately
    - Nested parenthesis evaluated inside out
  - Precedence rule:
    - There is an evaluation order in operators
      - Unary +, -
      - *, / , %
      - Binary +, -
  - Associativity rule:
    - Operators in the same sub-expression and at the same precedence level
      - Unary: right to left
      - Binary: left to right

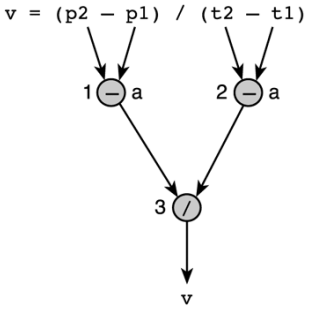
Evaluation Tree for area = PI * radius * radius;



Step-by-Step Expression Evaluation

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

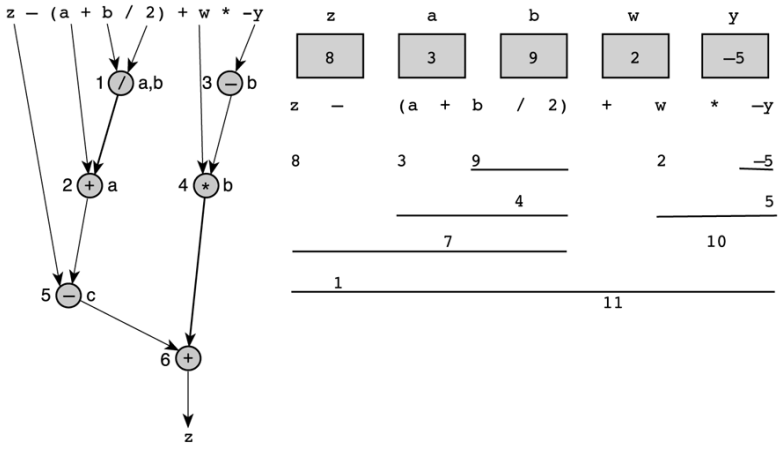

Evaluation for v = (p2 - p1) / (t2 - t1);




```
p1        p2        t1        t2
4.5       9.0       0.0       60.0

v =   (p2  –  p1)   /   (t2  –  t1)
      9.0     4.5       60.0     0.0
        4.5                  60.0
                0.075
```

Evaluation for z - (a + b / 2) + w * -y



Writing Mathematical Formulas

- Use parentheses as needed to specify the order of evaluation
  - Place numerator and denominator of a division in parentheses

    m = (a - b) / (c + d)
- Use extra parentheses for readability

  (a * b * c) + (d / e) – (a + b)
- Do not skip * as in mathematical formulas
  - In math: $d = b^2 – 4ac$
  - In C: d = b * b - 4 * a * c;
- Two operators can be one after the other

  a * -(b + c)

Case Study: Coin Processor

- Problem requirements
  - Convert change to personalized credit slip
  - User enters the number of each kind of coin
- Analyze the problem
  - Personalizing the slip: use customers initials
  - Count each type of coin
  - Total value of the coins in dollars and cents
  - Input data
    - Initials: first, middle, last are characters
    - Counts: dollars, quarters, dimes, nickels, pennies are integers
  - Output data
    - Dollars and cents: total_dollars and change are integers
  - Intermediate data
    - Total value in cents: total_cents is integer
  - Relationships
    - total_value =
    - total_dollars =
    - change =

Case Study: Coin Processor

- Design an algorithm
  1. Get and display the customer's initials
  2. Get the count for each kind of coin
  3. Compute the total value in cents
  4. Find the value in dollars and cents
  5. Display dollars and cents
  - Some steps need to refine!...
- Implement the algorithm
  - In the next slide
- Test and verify the program
- Maintain and update the program

Supermarket Coin Value Program

```
/*
 * Determines the value of a collection of coins.
 */
#include <stdio.h>
int
main(void)
{
     char first, middle, last;/ * input - 3 initials           */
     int pennies, nickels;  /* input - count of each coin type */
     int dimes, quarters;   /* input - count of each coin type */
     int dollars;           /* input - count of each coin type */
     int change;               /* output - change amount       */
     int total_dollars;        /* output - dollar amount       */
     int total_cents;          /* total cents                  */

     /* Get and display the customer's initials. */
     printf("Type in 3 initials and press return> ");
     scanf("%c%c%c", &first, &middle, &last);
     printf("\n%c%c%c, please enter your coin information.\n",
            first, middle, last);

     /* Get the count of each kind of coin. */
     printf("Number of $ coins > ");
     scanf("%d", &dollars);
     printf("Number of quarters> ");
```

(continued)

Supermarket Coin Value Program (cont'd)

```
     scanf("%d", &quarters);
     printf("Number of dimes   > ");
     scanf("%d", &dimes);
     printf("Number of nickels > ");
     scanf("%d", &nickels);
     printf("Number of pennies > ");
     scanf("%d", &pennies);

     /* Compute the total value in cents. */
     total_cents = 100 * dollars +25 * quarters + 10 * dimes +
                   5 * nickels + pennies;

     /* Find the value in dollars and change. */
     dollars = total_cents / 100;
     change = total_cents % 100;

     /* Display the credit slip with value in dollars and change. */
     printf("\n\n%c%c%c Coin Credit\nDollars: %d\nChange:  %d cents\n",
            first, middle, last, dollars, change);

     return (0);
}
```

```
Type in 3 initials and press return> JRH
JRH, please enter your coin information.
Number of $ coins > 2
Number of quarters> 14
Number of dimes   > 12
Number of nickels > 25
Number of pennies > 131

JRH Coin Credit
Dollars: 9
Change:  26 cents
```

Case Study: Coin Processor

- Test and verify the program
  - Try the program for several inputs
  - Make sure that program runs correctly
- Maintain and update the program
  - Later!...

Output Formatting

- Default formatting
- User-defined format
  - int: %4d (%nd)
    - Field width
    - Right justified
    - - sign included in the count
    - C expands the field width if necessary
  - double: %6.2f (%n.mf)
    - Field width
    - Decimal places
    - Decimal point, minus sign included in the field width
      - Values between -99.99 to 999.99 for %6.2f
    - At least one digit before decimal point
    - Values are rounded if there are more decimal places
      - -9.536 becomes -9.54
  - Use %d or %.3f not to have leading blanks

Input and Output Redirection

- Interactive mode
- Batch mode
  - Input Redirection: standard input is associated with a file instead of keyboard

    myprog < inputfile
    - No need to display prompting message
    - Display the message about input (echo print)
  - Output Redirection: standard output is associated with a file instead of screen

    myprog > outputfile
    - Can print the file to get the hardcopy

Batch Version of Miles-to-Kilometers Conversion Program

```
1.  /* Converts distances from miles to kilometers.    */
2.
3.  #include <stdio.h>    /* printf, scanf definitions */
4.  #define KMS_PER_MILE 1.609 /* conversion constant  */
5.
6.  int
7.  main(void)
8.  {
9.        double miles,  /* distance in miles                                  */
10.             kms;    /* equivalent distance in kilometers                  */
11.
12.       /* Get and echo the distance in miles. */
13.       scanf("%lf", &miles);
14.       printf("The distance in miles is %.2f.\n", miles);
15.
16.       /* Convert the distance to kilometers. */
17.       kms = KMS_PER_MILE * miles;
18.
19.       /* Display the distance in kilometers. */
20.       printf("That equals %.2f kilometers.\n", kms);
21.
22.       return (0);
23. }

The distance in miles is 112.00.
That equals 180.21 kilometers.
```

Use of Input/Output Files

- C allows to explicitly name an input or output file
  - Declaring file pointer

    FILE * inp;

    FILE * outp;
  - Opening file

    inp = fopen(“filename”, “r”);

    outp = fopen(“filename”, “w”);
  - Reading from a file

    fscanf(inp, “%d”, &nickels);
  - Writing to a file

    fprintf(outp, “Total is %d \n”, value);
  - Closing file

    fclose(inp);

Miles-to-Kilometers Conversion Program with Named Files

```
1.  /* Converts distances from miles to kilometers.    */
2.
3.  #include <stdio.h>    /* printf, scanf, fprint, fscanf, fopen, fclose
4.                          definitions                 */
5.  #define KMS_PER_MILE 1.609 /* conversion constant  */
6.
7.  int
8.  main(void)
9.  {
10.       double miles, /* distance in miles                                    */
11.              kms;   /* equivalent distance in kilometers                   */
12.       FILE   *inp,  /* pointer to input file                               */
13.              *outp; /* pointer to output file                              */
14.
15.       /* Open the input and output files.    */
16.       inp = fopen("b:distance.dat", "r");
17.       outp = fopen("b:distance.out", "w");
18.
19.       /* Get and echo the distance in miles. */
20.       fscanf(inp, "%lf", &miles);
21.       fprintf(outp, "The distance in miles is %.2f.\n", miles);
22.
23.       /* Convert the distance to kilometers. */
24.       kms = KMS_PER_MILE * miles;
25.
26.       /* Display the distance in kilometers. */
27.       fprintf(outp, "That equals %.2f kilometers.\n", kms);
28.
29.       /* Close files. */
30.       fclose(inp);
31.       fclose(outp);
32.
33.       return (0);
34. }
```

Contents of input file `distance.dat`

```
112.0
```

Contents of output file `distance.out`

```
The distance in miles is 112.00.
That equals 180.21 kilometers.
```

Case Studies:

- Compute change for a given amount of money

Programming Errors

- Error = bug
- Process of correcting errors: debugging
- Error messages
  - Depends on the system used
  - Not always easy to understand
- Three kind of errors:
  - Syntax errors
    - Violation of grammar rule
    - Detected by the compiler
  - Run-time errors
    - Detected while execution
    - Illegal operation, division by zero etc.
  - Logic errors
    - Program runs but produces incorrect result

A Program with Syntax Errors

```
221 /* Converts distances from miles to kilometers. */
222
223 #include <stdio.h>         /* printf, scanf definitions   */
266 #define KMS_PER_MILE 1.609 /* conversion constant         */
267
268 int
269 main(void)
270 {
271       double kms
272
273       /* Get the distance in miles. */
274       printf("Enter the distance in miles> ");
***** Semicolon added at the end of the previous source line

275       scanf("%lf", &miles);
***** Identifier "miles" is not declared within this scope
***** Invalid operand of address-of operator

276
277       /* Convert the distance to kilometers. */
278       kms = KMS_PER_MILE * miles;
***** Identifier "miles" is not declared within this scope

279
280       /* Display the distance in kilometers. * /
281       printf("That equals %f kilometers.\n", kms);
282
283       return (0);
284 }
***** Unexpected end-of-file encountered in a comment
***** "}" inserted before end-of-file
```

A Program with a Run-Time Error

```
111 #include <stdio.h>
262
263 int
264 main(void)
265 {
266       int    first, second;
267       double temp, ans;
268
269       printf("Enter two integers> ");
270       scanf("%d%d", &first, &second);
271       temp = second / first;
272       ans = first / temp;
273       printf("The result is %.3f\n", ans);
274
275       return (0);
276 }

Enter two integers> 14 3
Arithmetic fault, divide by zero at line 272 of routine main
```

Revised Coin Value Program

```
1.  int
2.  main(void)
3.  {
4.       char first, middle, last; /* input - 3 initials          */
5.       int pennies, nickels;  /* input - count of each coin type */
6.       int dimes, quarters;   /* input - count of each coin type */
7.       int dollars;           /* input - count of each coin type */
8.       int change;               /* output - change amount      */
9.       int total_dollars;        /* output - dollar amount      */
10.      int total_cents;          /* total cents                 */
11.      int year;                 /* input – year                */
12.
13.      /* Get the current year.                                  */
14.      printf("Enter the current year and press return> ");
15.      scanf("%d", &year);
16.
17.      /* Get and display the customer's initials.               */
18.      printf("Type in 3 initials and press return> ");
19.      scanf("%c%c%c", &first, &middle, &last);
20.      printf("\n%c%c%c, please enter your coin information for %d.\n",
21.             first, middle, last, year);
         ...
```


A Program That Produces Incorrect Results

```
1.  #include <stdio.h>
2.
3.  int
4.  main(void)
5.  {
6.        int    first, second, sum;
7.
8.        printf("Enter two integers> ");
9.        scanf("%d%d", first, second); /* ERROR!! should be  &first, &second */
10.       sum = first + second;
11.       printf("%d + %d = %d\n", first, second, sum);
12.
13.       return (0);
14. }

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