ELE108
Introduction to Programming

Dr. Ali Ziya Alkar

Dr. Mehmet Demirer

Outline

- Overview of C
- General form of a C program
- C Language Elements

History & Philosophy

- C is developed in 1972 by Dennis Ritchie at the AT&T Bell Laboratories for use with the Unix.
- C is a minimalistic programming language.
- The most commonly used programming language for writing system software.
- Machine independent: by minimal change in source code, can be compiled in a wide variety of platform and operating system.

Why C?

- Many, many companies/research projects do all their programming in C.
- Small, compact code.
- Produces optimized programs that runs faster.
- Low-level access to computer memory via machine addresses and pointers.
- Low level (BitWise) programming readily available.
- Can be compiled on a variety of computers.

What's Missing?

- No Objects.
- Poor error detection which can make it difficult to use for the beginner
  - No automatic garbage collection.
  - No bounds checking of arrays and allocated memory segments.
  - No exception handling.
- No native support for multithreading and networking, though these facilities are provided by popular libraries
- No standard libraries for graphics and several other application programming needs

A Simple, Example C Program

```
/* helloworld.c */
#include <stdio.h>
int main(void) {
  printf("Hello World!\n");
  return(0);
}
```

- Every C program has a `main` function. It is very much the same as the `main` method in a Java class.
- `printf` is also the name of a function. It can do much the same as Java's `System.out.print`.
- This program can use the `printf` function, because of the line `#include <stdio.h>` in the source code. This line is similar to Java's `import java.io.*`

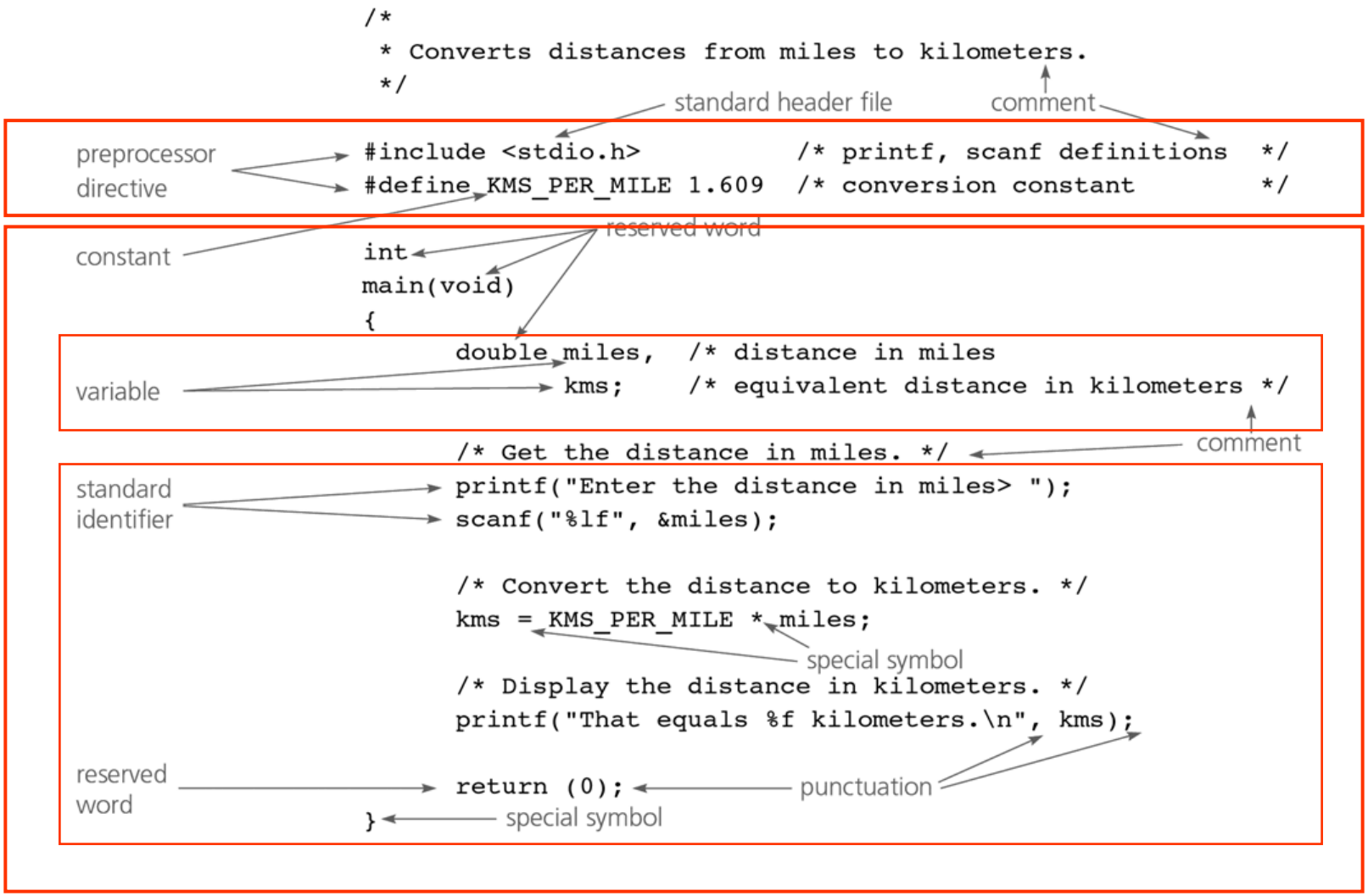
General Form of a C program

```
preprocessor directives
main function heading
{
    declarations
    executable statements
}
```

- Preprocessor directives are instructions to C Preprocessor to modify The text of a C program before compilation.
- Every variable has to be declared first.

- Executable statements are translated into machine language and eventually executed.
- Executable statements perform computations on the declared variables or input/output operations.

Miles to Kilometers conversion



C Language Elements

- Preprocessor Directives
- Comments
- The “main” function
- Variable Declarations and Data Types
- Executable Statements
- Reserved Words
- Identifiers

Preprocessor Directives

```
/* Converts distances from miles to kilometers */
#include <stdio.h>                    /* printf, scanf definitions */
#define KMS_PER_MILE 1.609            /* conversion constant */

int main(void)
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

Preprocessor Directives

- Preprocessor directives are commands that give instructions to the C preprocessor.
- Preprocessor is a system program that modifies a C program prior to its compilation.
- Preprocessor directives begins with a #
  - Example. #include or #define

#include

- `#include` is used to include other source files into your source file.
- The `#include` directive gives a program access to a library.
- **Libraries** are useful functions and symbols that are predefined by the C language (standard libraries).
  - Example: You must include `stdio.h` if you want to use the `printf` and `scanf` library functions.
  - `# include<stdio.h>` insert their definitions to your program before compilation.

#define

- The `#define` directive instructs the preprocessor to replace each occurrence of a text by a particular constant value before compilation.
- `#define` replaces all occurrences of the text you specify with value you specify
  - Example:

```
#define KMS_PER_MILES 1.60
#define PI 3.14159
```

Comments

```
/* Converts distances from miles to kilometers */

#include <stdio.h>                      /* printf, scanf definitions */
#define KMS_PER_MILE 1.609              /* conversion constant */

int main(void)
{
    double miles,       //distance in miles
           kms;         //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

Comments

- Comments provide supplementary information making it easier for us to understand the program, but are ignored by the C compiler.
- Two forms of comments:
  - /* */ - anything between them with be considered a comment, even if they span multiple lines.
  - // - anything after this and before the end of the line is considered a comment.
- Comments are used to create **Program Documentation**
  - Information that help others read and understand the program.
- The start of the program should consist of a comment that includes programmer's name, date of the current version, and a brief description of what the program does.
- **Always Comment your Code!**

The “main” Function

```
/* Converts distances from miles to kilometers */
#include <stdio.h>                      /* printf, scanf definitions */
#define KMS_PER_MILE 1.609              /* conversion constant */
int main(void)
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

The “main” Function

- The heading `int main(void)` marks the beginning of the `main` function where program execution begins.
- Every C program has a `main` function.
- Braces ({,}) mark the beginning and end of the body of function main.
- A function body has to parts:
  - **declarations** - tell the compiler what memory cells are needed in the function
  - **executable statements** - (derived from the algorithm) are translated into machine language and later executed by the compiler.

Variables and Data Types

```
/* Converts distances from miles to kilometers */
#include <stdio.h>                      /* printf, scanf definitions */
#define KMS_PER_MILE 1.609              /* conversion constant */
int main(void)
{
    double miles,       //distance in miles
           kms;         //equivalent distance in kilometers
    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);
    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;
    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);
    return (0);
}
```

Variables Declarations

- **Variable** – The memory cell used for storing a program's data and its computational results
  - Variable's value can change.
  - Example: `miles`, `kms`
- **Variable declarations** –Statements that communicates to the compiler the names of variables in the program and the kind of information they can store.
  - Example: `double miles`
    - Tells the compiler to create space for a variable of type `double` in memory with the name `miles`.
  - C requires you to declare every variable used in the program.

Data Types

- **Data Types**: a set of values and a set of operations that can be performed on those values
  - **int**: Stores integer values – whole numbers
    - 65, -12345
  - **double**: Stores real numbers – numbers that use a decimal point.
    - 3.14159 or 1.23e5 (which equals 123000.0)
  - **char**: An individual character value.
    - Each char value is enclosed in single quotes. E.g. ‘A’, ‘*’.
    - Can be a letter, a digit, or a special symbol
  - Arithmetic operations (+, -, *, /) and compare can be performed in case of **int** and **double**. Compare can be performed in **char** data.

Executable Statements

```
/* Converts distances from miles to kilometers */

#include <stdio.h>                      /* printf, scanf definitions */
#define KMS_PER_MILE 1.609              /* conversion constant */

int main(void)
{
    double miles,       //distance in miles
           kms;         //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

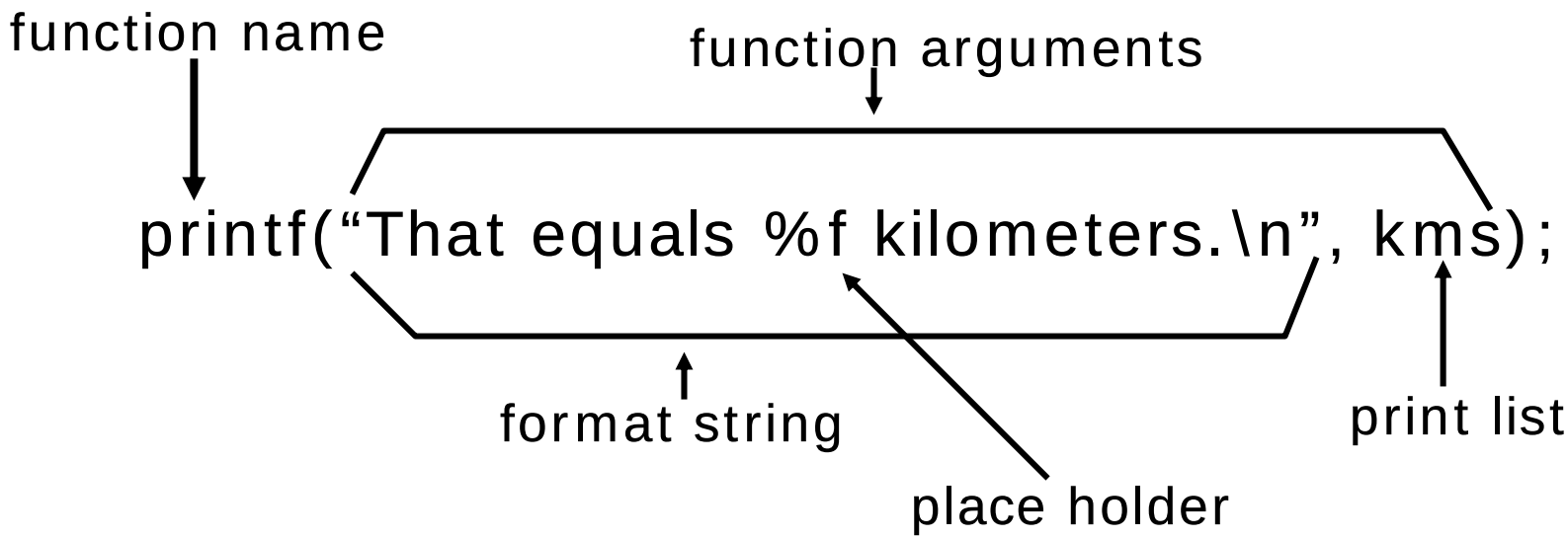
Executable Statements

- Executable Statements: C statements used to write or code the algorithm. C compiler translates the executable statements to machine code.
  - Input/Output Operations and Functions
    - `printf` Function
    - `scanf` Function
  - Assignment Statements
  - return Statement

Input/Output Operations and Functions

- **Input operation -** data transfer from the outside world into computer memory
- **Output operation -** program results can be displayed to the program user
- **Input/output functions -** special program units that do all input/output operations
  - `printf` = output function
  - `scanf` = input function
- **Function call -** in C a function call is used to call or activate a function
  - Calling a function means asking another piece of code to do some work for you

The printf Function



Placeholders

- Placeholder always begins with the symbol %
  - It marks the place in a format string where a value will be printed out or will be inputed (in this case, kms)
- Format strings can have multiple placeholders, if you are printing multiple values

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

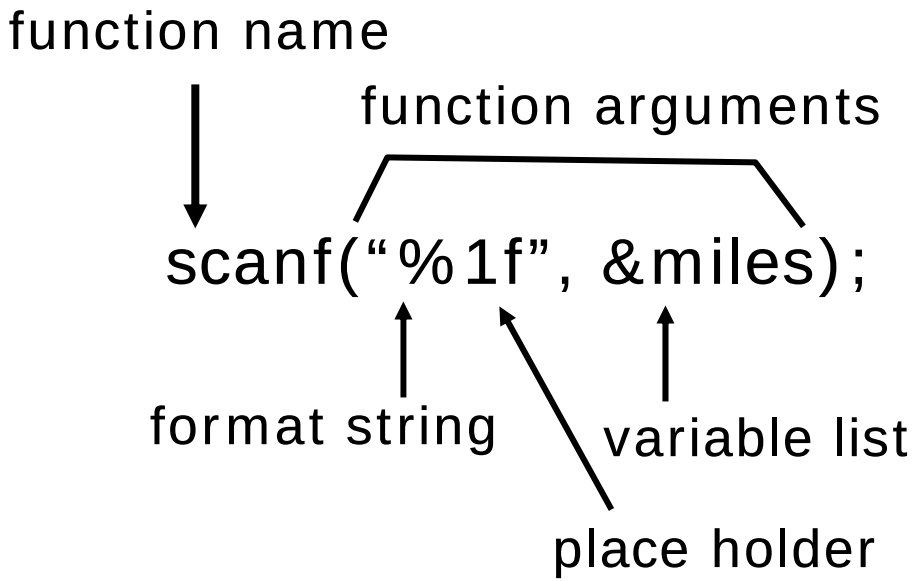
- newline escape sequence – ‘\n’ terminates the current line

Displaying Prompts

- When input data is needed in an interactive program, you should use the `printf` function to display a **prompting message**, or **prompt**, that tells the user what data to enter.

```
Printf(“Enter the distance in miles> “);
```

The scanf Function

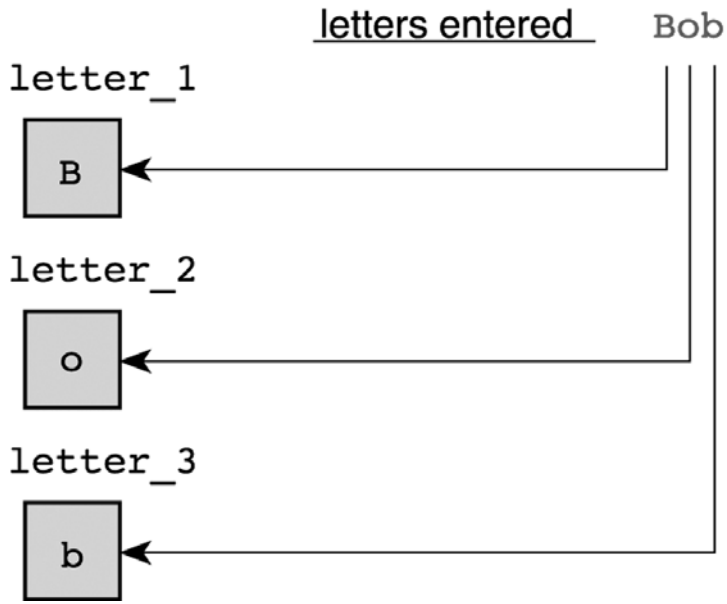


- When user inputs a value, it is stored in variable `miles`.
- The placeholder type tells the function what kind of data to store into variable miles.

- The & is the C address of operator. The & operator in front of variable `miles` tells the `scanf` function the location of variable miles in memory.

Fig 2.6: Scanning data line Bob

```
char letter_1, letter_2, letter_3;
….
Scanf(“%c%c%c”, &letter_1, &letter_2, &letter_3);
```



Assignment Statements

- **Assignment statement -** Stores a value or a computational result in a variable

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

- The assignment statement above assigns a value to the variable `kms`. The value assigned is the result of the multiplication of the constant `KMS_PER_MILE` by the variable `miles`.

Figure 2.3 Effect of kms = KMS_PER_MILE * miles;

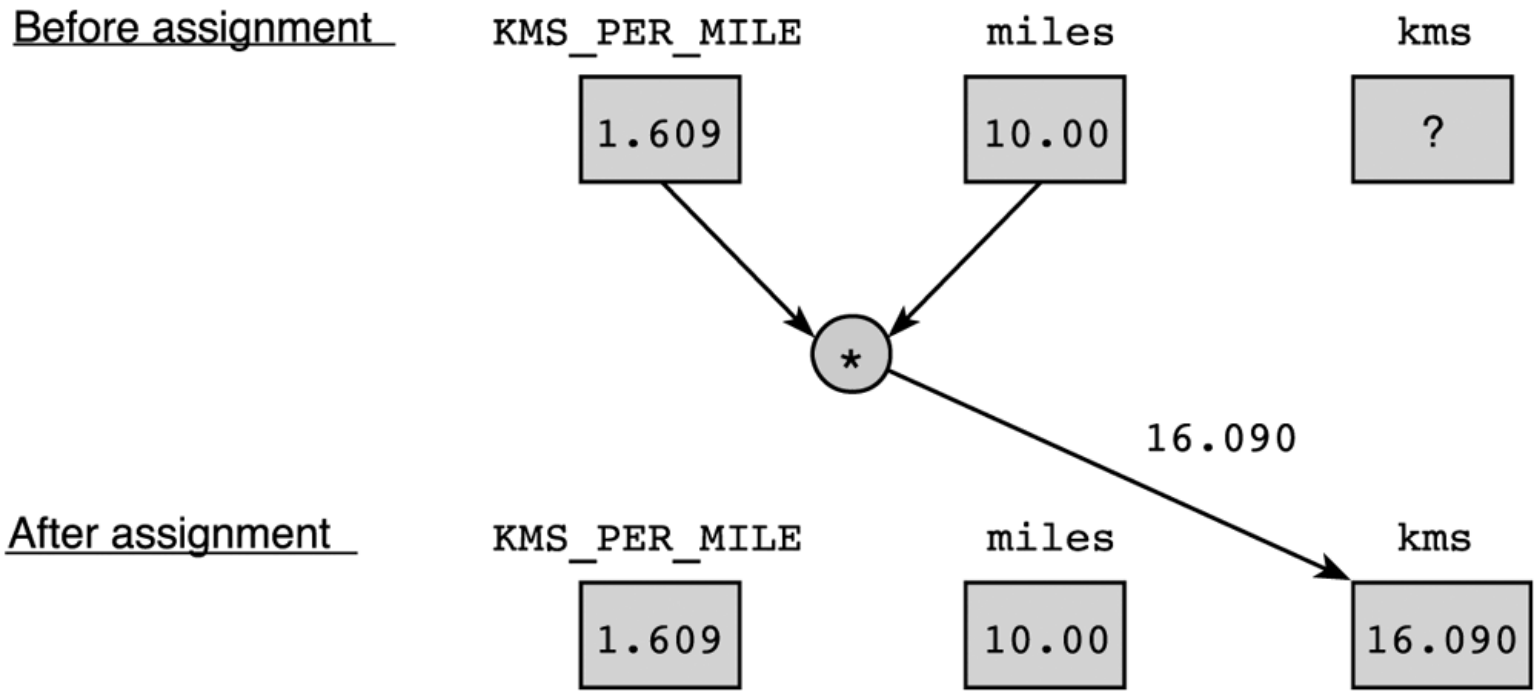
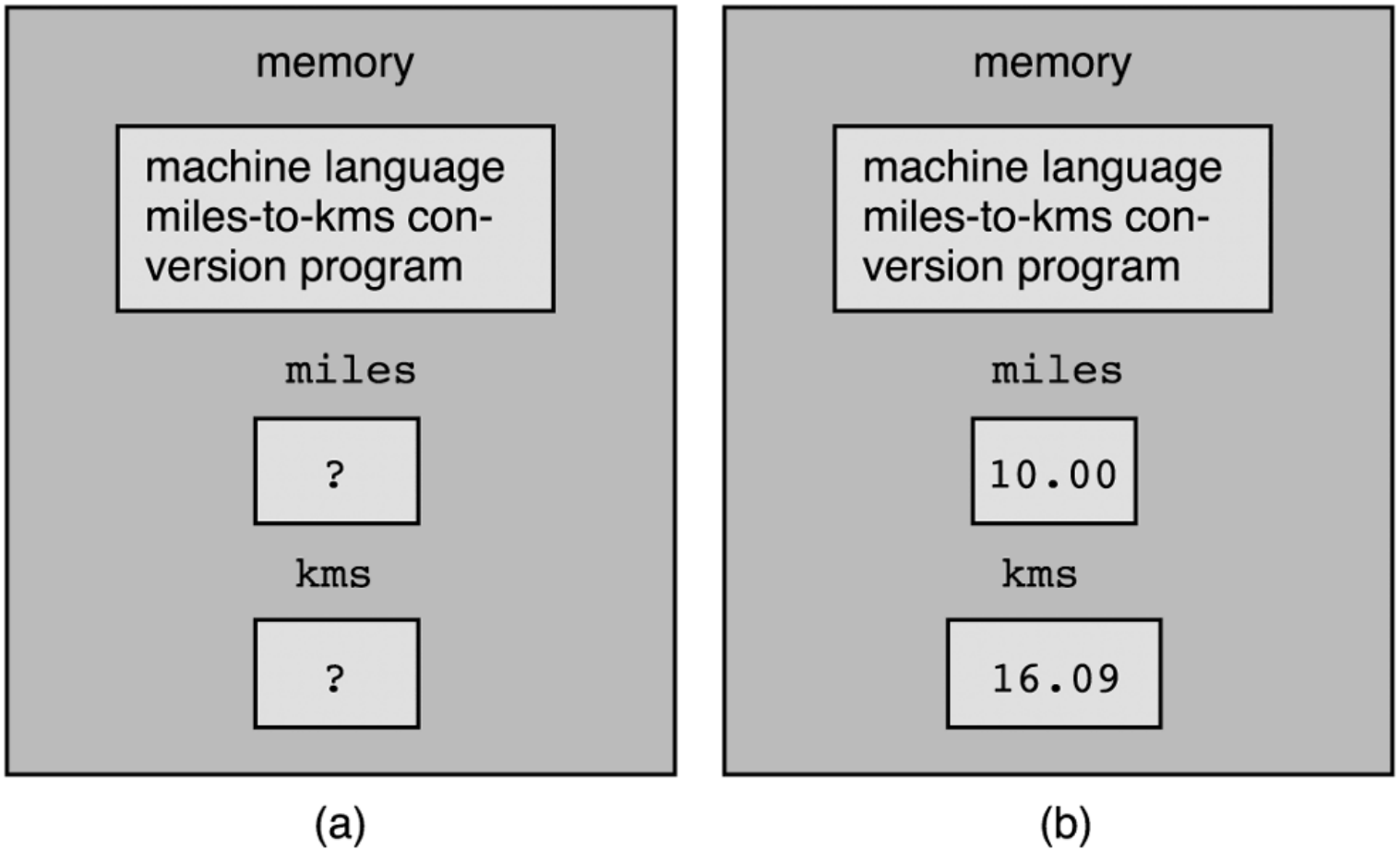


Figure 2.2 Memory(a) Before and (b) After Execution of a Program



More on Assignments

- In C the symbol = is the assignment operator
  - Read it as ”becomes”, ”gets”, or ”takes the value of” rather than ”equals” because it is not equivalent to the equal sign of mathematics. In C, == tests equality.
- In C you can write assignment statements of the form:

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

  where the variable sum appears on both sides of the assignment operator.

  This is obviously not an algebraic equation, but it illustrates a common programming practice. This statement instructs the computer to add the current value of `sum` to the value of `item`; the result is then stored back into `sum`.

return Statement

```
return (0);
```

- Transfers control from your program to the operating system.
- `return (0)` returns a 0 to the Operating System and indicates that the program executed without error.
- It does not mean the program did what it was suppose to do. It only means there were no syntax errors. There still may have been logical errors.
- Once you start writing your own functions, you'll use the `return` statement to return information to the caller of the function.

Reserved Words

```
/* Converts distances from miles to kilometers */
#include <stdio.h>                    /* printf, scanf definitions */
#define KMS_PER_MILE 1.609            /* conversion constant */
int main(void)
{
    double miles,      //distance in miles
           kms;        //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

Reserved words

- A word that has special meaning to C and can not be used for other purposes.
- These are words that C reserves for its own uses (declaring variables, control flow, etc.)
  - For example, you couldn't have a variable named `return`
- Always lower case
- Appendix E has a list of them all (ex: `double`, `int`, `if` , `else`, …)

Identifiers

```
/* Converts distances from miles to kilometers */
#include <stdio.h>                    /* printf, scanf definitions */
#define KMS_PER_MILE 1.609            /* conversion constant */
int main(void)
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

Standard Identifiers

- **Identifier -** A name given to a variable or an operation
  - In other words, Function names and Variable names
- **Standard Identifier -** An identifier that is defined in the standard C libraries and has special meaning in C.
  - Example: `printf, scanf`
  - Standard identifiers are not like reserved words; you could redefine them if you want to. But it is not recommended.
    - For example, if you create your own function called `printf`, then you may not be able to access the library version of `printf`.

User Defined Identifiers

- We choose our own identifiers to name memory cells that will hold data and program results and to name operations that we define (more on this in Chapter 3).
- **Rules for Naming Identifiers:**
  - An identifier must consist only of letters, digits, and underscores.
  - An identifier cannot begin with a digit.
  - A C reserved word cannot be used as an identifier.
  - A standard identifier should not be redefined.
- Valid identifiers: `letter1, inches, KM_PER_MILE`
- Invalid identifiers: `1letter, Happy*trout, return`

Few Guidelines for Naming Identifiers

- Some compliers will only see the first 31 characters of the identifier name, so avoid longer identifiers
- Uppercase and lowercase are different
  - `LETTER` != `Letter` != `letter`
  - Avoid names that only differ by case; they can lead to hard to find bugs
- Choose meaningful identifiers that are easy to understand. Example: `distance = rate * time` means a lot more than `x=y*z`
- All uppercase is usually used for constant macros (`#define`)
  - `KMS_PER_MILE` is a defined constant
  - As a variable, we would probably name it `KmsPerMile` or `Kms_Per_Mile`

Punctuation and Special Symbols

```
/* Converts distances from miles to kilometers */
#include <stdio.h>                    /* printf, scanf definitions */
#define KMS_PER_MILE 1.609            /* conversion constant */
int main(void)
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

Punctuation and Special Symbols

- **Semicolons (;)** – Mark the end of a statement
- **Curly Braces ({,}) –** Mark the beginning and end of the main function
- **Mathematical Symbols (*,=) –** Are used to assign and compute values