ICS103 Programming in C

Lecture 3: Introduction to C (2)

Outline

- Overview of C
  - History & Philosophy
  - Why C?
  - What's Missing?
- General form of a C program
- C Language Elements
  - Preprocessor Directives
  - Comments
  - The "main" function
  - Variable Declarations and Data Types
  - Executable Statements
  - Reserved Words
  - Identifiers
  - Formatting Numbers in Program Output

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);
}
```

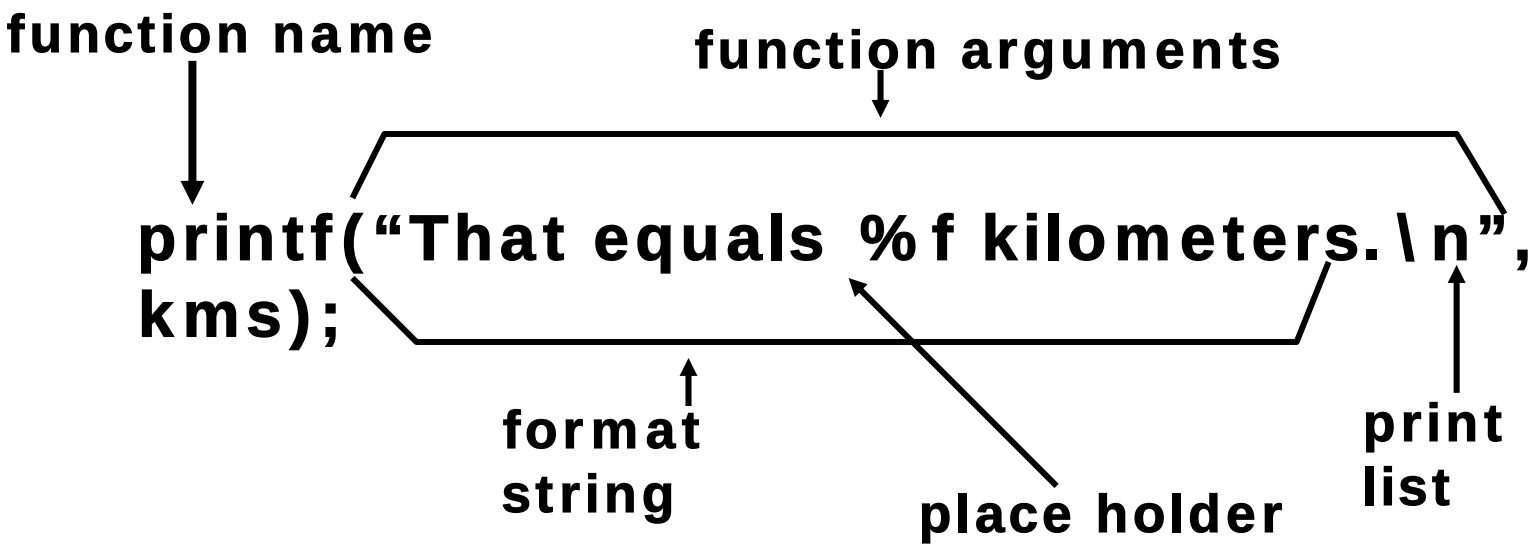
Executable Statements

- 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`

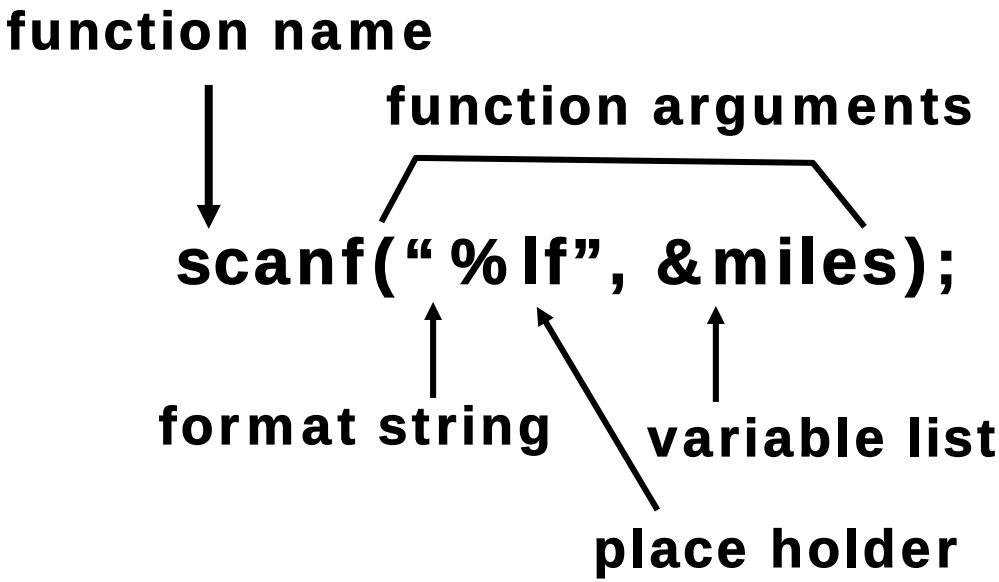
- newline escape sequence – ‘\n’ terminates the current line

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

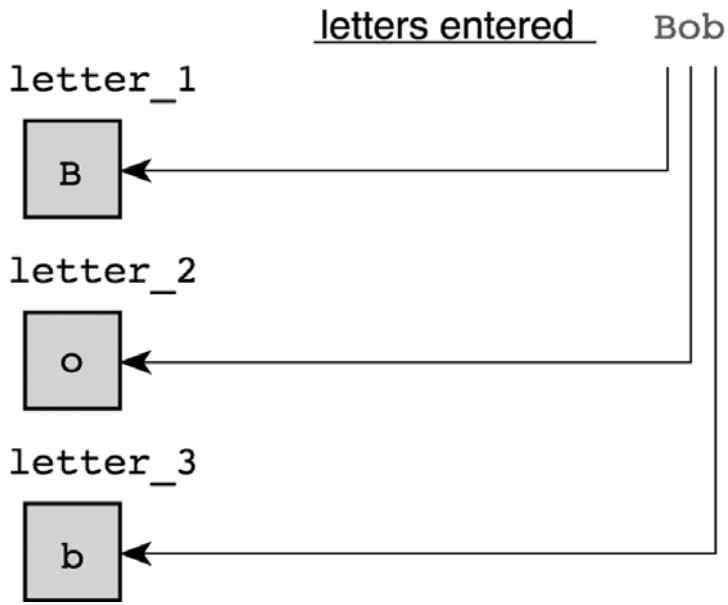


- When user inputs a value, it is stored in variable `miles`.
- The placeholder type tells the function what kind of data to store into variable miles.

- 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;

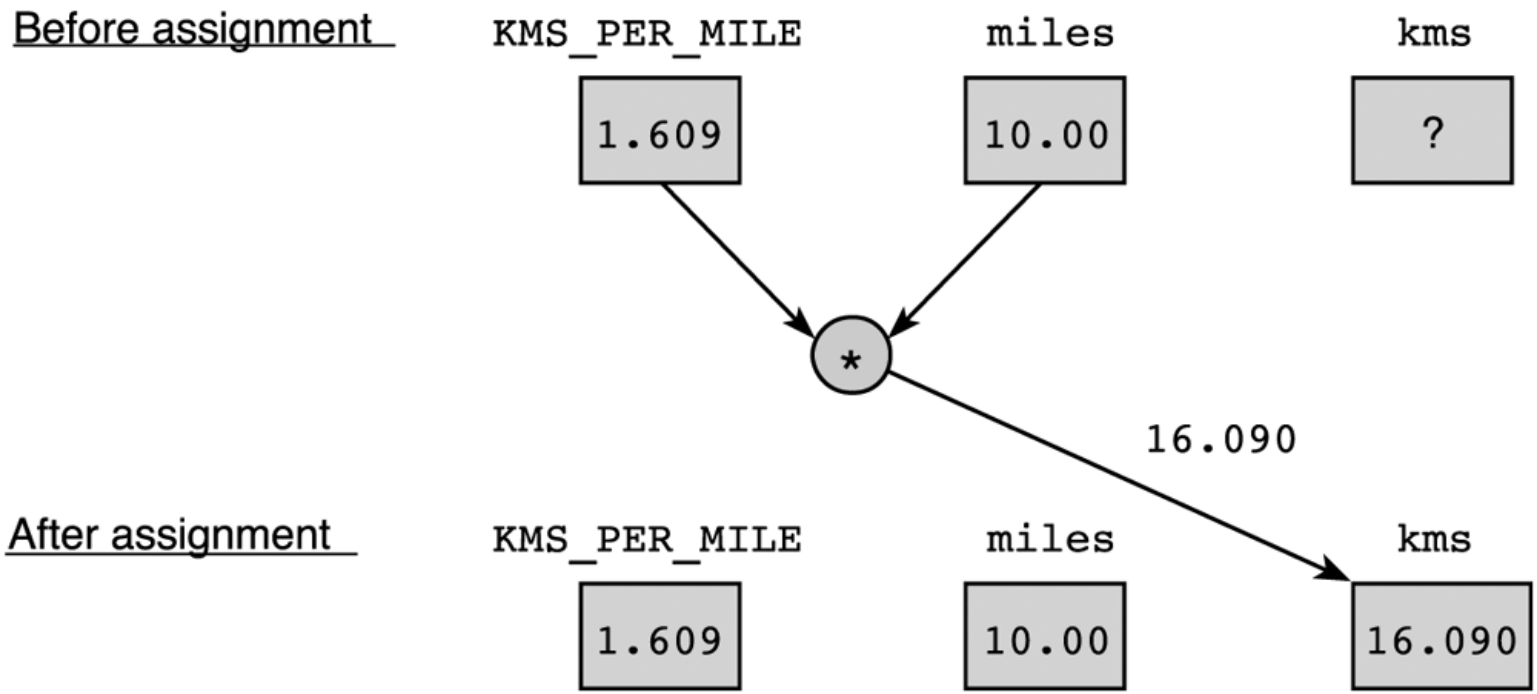
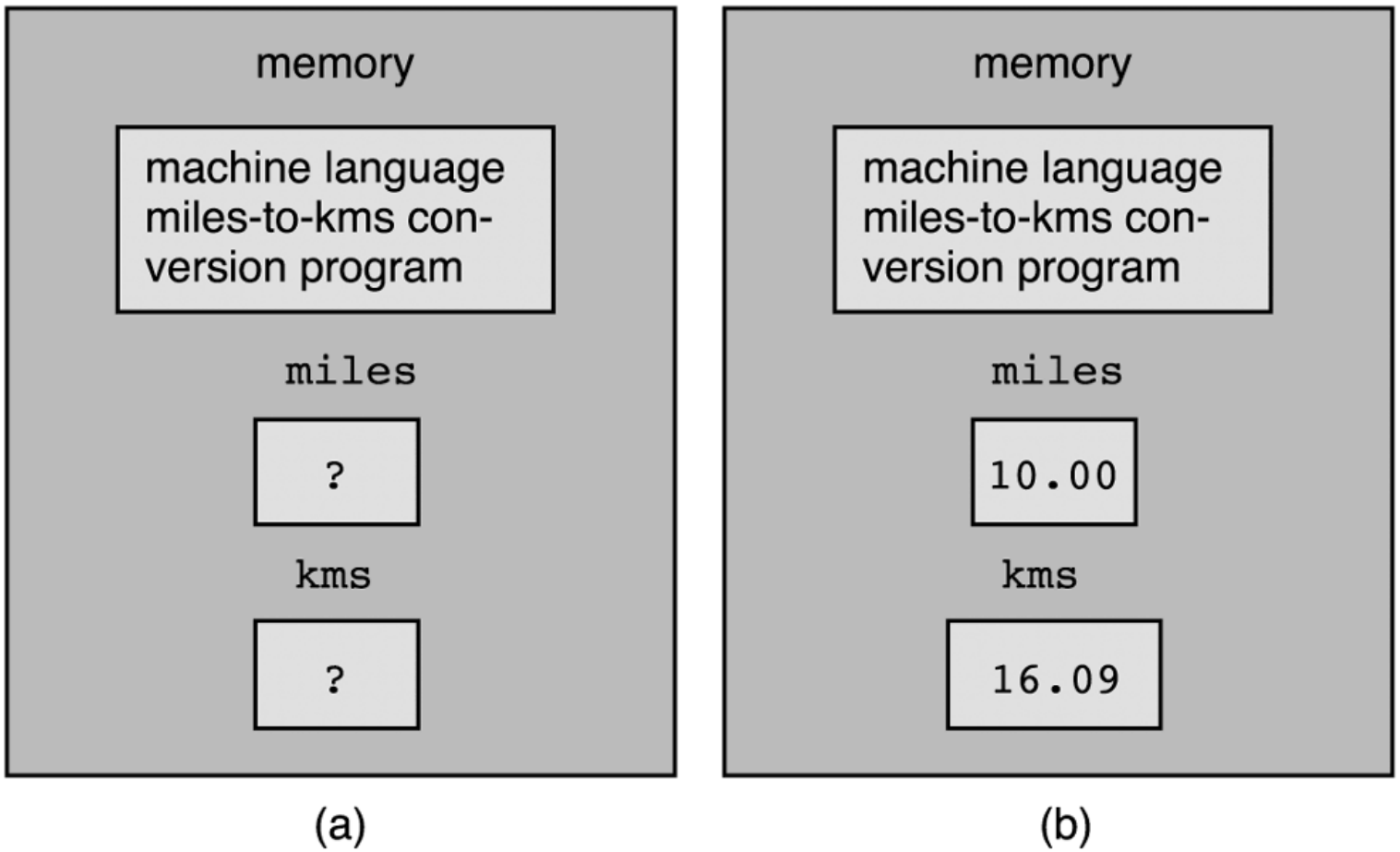


Figure 2.2 Memory(a) Before and (b) After Execution of a Program



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 H 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 problems 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

Formatting Numbers in Program Output (for integers)

- You can specify how printf will display numeric values
- Use d for integers. %#d
  - % - start of placeholder
  - # - field width (optional) – the number of columns to use to display the output.
  - d - placeholder for integers

```
int n = 123;
printf("%1d\n", n);
printf("%3d\n", n);
printf("%4d\n", n);
```

```
123
123
 123
```

Formatting Numbers in Program Output (for double)

- Use %n.mf for double
  - % - start of placeholder
  - n - field width (optional)
  - m – Number of decimal places (optional)
  - f - placeholder for real numbers

```
double n = 123.456;
printf("%8.0f\n", n);          123
printf("%8.2f\n", n);       123.46
printf("%8.3f\n", n);      123.456
printf("%8.4f\n", n);     123.4560
Printf("%.2f\n", n);      123.46
```