Variables in C

Topics

- What is Variable
- Naming Variables
- Declaring Variables
- Using Variables
- The Assignment Statement

What Are Variables in C?

- Variables are the names that refer to sections of memory into which **data can be stored**.
- **Variables** in C have the same meaning as variables in algebra. That is, they represent some unknown, or variable, value.

$$x = a + b$$

$$z + 2 = 3(y - 5)$$

- Remember that variables in algebra are represented by a single alphabetic character.

Naming Variables

- Rules for variable naming:
  - Can be composed of letters (both uppercase and lowercase letters), digits and underscore only.
  - The first character should be either a letter or an underscore(not any digit).
  - Punctuation and special characters are not allowed except underscore.
  - Variable name should not be keywords.
  - names are case sensitive.
  - There is no rule for the length of a variable name. However, the first 31 characters are discriminated by the compiler. So, the first 31 letters of two name in a program should be different.

Reserved Words (Keywords) in C

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

volatile while

Naming Conventions

- C programmers generally agree on the following **conventions** for naming variables.
  - Begin variable names with lowercase letters
  - Use meaningful identifiers
  - Separate "words" within identifiers with underscores or mixed upper and lower case.
  - Examples: surfaceArea surface_Area surface_area
  - Be consistent!

Naming Conventions (con’t)

- Use all uppercase for **symbolic constants** (used in **#define** preprocessor directives).
- Examples:

```
#define PI 3.14159
#define AGE  52
```

Case Sensitivity

- C is **case sensitive**
  - It matters whether an **identifier**, such as a variable name, is uppercase or lowercase.
  - Example:

    area

    Area

    AREA

    ArEa

    are all seen as <u>different</u> variables by the compiler.

Which Are Legal Identifiers?

AREA

3D

Last-Chance

x_yt3

num$

lucky***

area_under_the_curve

num45

#values

pi

%done

Declaring Variables

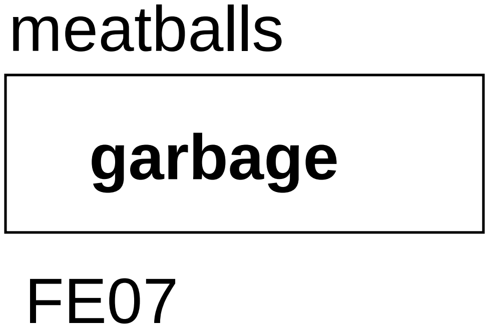
- Before using a variable, you must give the compiler some information about the variable; i.e., you must **declare** it.
- The **declaration statement** includes the **data type** of the variable.
- Examples of variable declarations:

```
int  meatballs ;
float  area ;
```

Declaring Variables (con’t)

- When we declare a variable
  - Space is set aside in memory to hold a value of the specified data type
  - That space is associated with the variable name
  - That space is associated with a unique **address**
- Visualization of the declaration

  int meatballs ;



meatballs

garbage

FE07

More About Variables

C has three basic predefined data types:

- Integers (whole numbers)
  - **Int**
- Floating point (real numbers)
  - **float**,
  - **double**
- Characters
  - **char**

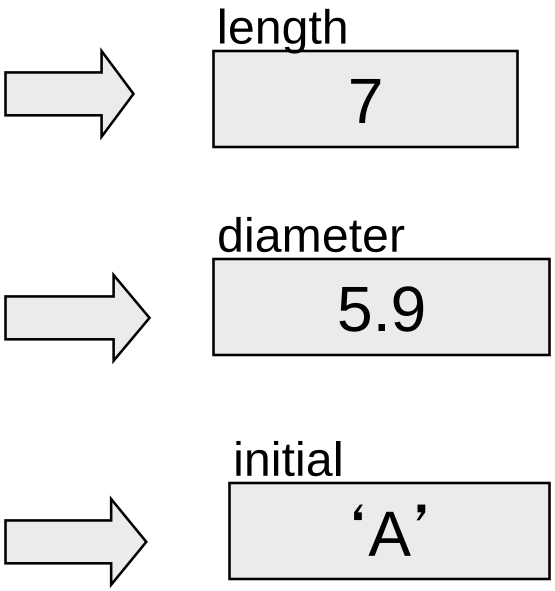
Using Variables: Initialization

- Variables may be given initial values, or **initialized**, when declared. Examples:

```
int length = 7 ;

float diameter = 5.9 ;

char initial = ‘A’ ;
```



Using Variables: Initialization (con’t)

- Do not “hide” the initialization
  - put initialized variables on a separate line
  - a comment is always a good idea
  - Example:

```
int height ;         /* rectangle height */
int width = 6 ;   /* rectangle width   */
int area ;           /* rectangle area     */
```

NOT int height, width = 6, area ;

Using Variables: Assignment

- Variables may have values assigned to them through the use of an **assignment statement**.
- Such a statement uses the **assignment operator =**
- This operator <u>does not</u> denote equality. It assigns the value of the righthand side of the statement (the **expression**) to the variable on the lefthand side.
- Examples:

```
diameter = 5.9 ;
area = length * width ;
```

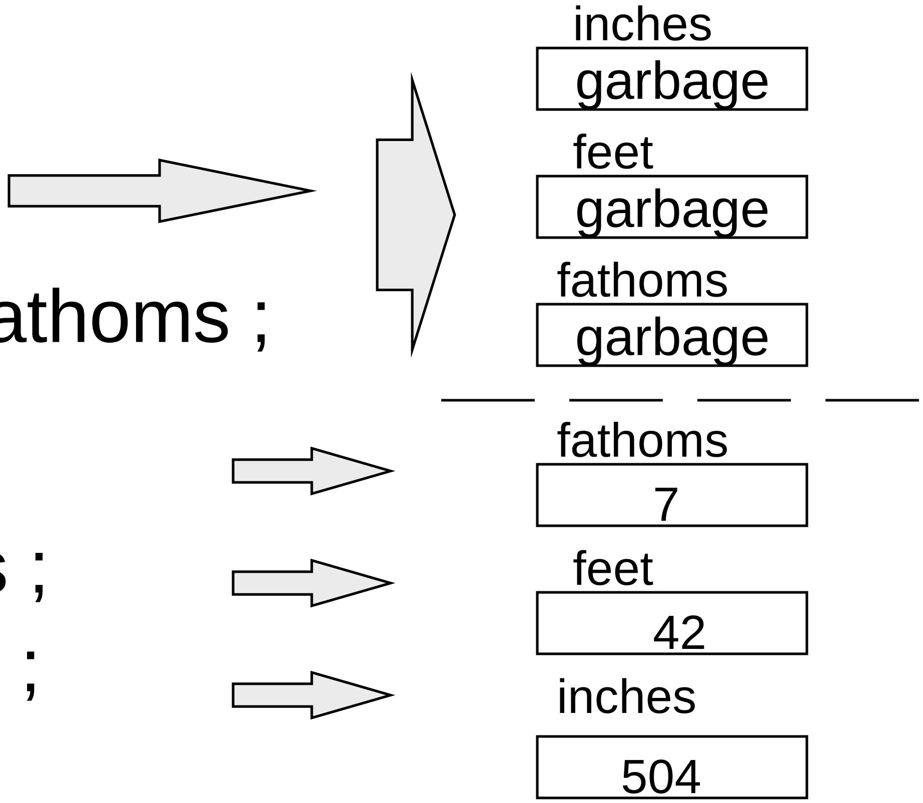
Note that only single variables may appear on the lefthand side of the assignment operator.

Example: Declarations and Assignments

```
#include <stdio.h>

int main( )
{
      int  inches, feet, fathoms ;

      fathoms = 7 ;
      feet = 6 * fathoms ;
      inches = 12 * feet ;
            •
            •
            •
```



inches

garbage

feet

garbage

fathoms

garbage

fathoms

7

feet

42

inches

504

Example: Declarations and Assignments (cont’d)

```
        •
        •
        •
    printf (“Its depth at sea: \n”) ;
    printf (“    %d fathoms \n”, fathoms) ;
    printf (“    %d feet \n”, feet) ;
    printf (“    %d inches \n”, inches) ;

    return 0 ;
}
```

Enhancing Our Example

- What if the depth were really 5.75 fathoms? Our program, as it is, couldn’t handle it.
- Unlike integers, floating point numbers can contain decimal portions. So, let’s use floating point, rather than integer.
- Let’s also ask the user to enter the number of fathoms, rather than **“hard-coding”** it in.

Enhanced Program

```
#include <stdio.h>
int main ( )
{
    float  inches, feet, fathoms ;
    printf (“Enter the depth in fathoms : ”) ;
    scanf (“%f”, &fathoms) ;
    feet = 6 * fathoms ;
    inches = 12 * feet ;
    printf (“Its depth at sea: \n”) ;
    printf (“    %f fathoms \n”, fathoms) ;
    printf (“    %f feet \n”, feet) ;
    printf (“    %f inches \n”, inches) ;
    return 0 ;
}
```

Final “Clean” Program

```
#include <stdio.h>
int main( )
{
   float  inches ;       /* number of inches deep   */
   float feet ;          /* number of feet deep     */
   float fathoms ;      /* number of fathoms deep */

   /* Get the depth in fathoms from the user */

   printf (“Enter the depth in fathoms : ”) ;
   scanf (“%f”, &fathoms) ;

   /* Convert the depth to inches */

   feet = 6 * fathoms ;
   inches = 12 * feet ;
```

Final “Clean” Program (con’t)

```
    /* Display the results */

    printf (“Its depth at sea: \n”) ;
    printf (“    %f fathoms \n”, fathoms) ;
    printf (“    %f feet \n”, feet);
    printf (“    %f inches \n”, inches);

    return 0 ;
}
```

Good Programming Practices

- Place each variable declaration on its own line with a descriptive comment.
- Place a comment before each logical “chunk” of code describing what it does.
- Do not place a comment on the same line as code (with the exception of variable declarations).
- Use spaces around all arithmetic and assignment operators.
- Use blank lines to enhance readability.

Good Programming Practices (con’t)

- Place a blank line between the last variable declaration and the first executable statement of the program.
- Indent the body of the program 3 to 4 tab stops -- be consistent!