Variables in C

CMSC 104, Spring 2014

Christopher S. Marron

(thanks to John Park for slides)

Variables in C

Topics

- Naming Variables
- Declaring Variables
- Using Variables
- The Assignment Statement

What Are Variables in C?

- **Variables** in C have a similar meaning as variables in algebra. That is, they represent some unknown, or variable, value.

$$x = a + b$$

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

- Variables in algebra are typically represented by a single alphabetic character.

Legal Identifiers in C

- Variables in C are also called **identifiers.**
- Variables in C may be given names containing multiple characters.
- Legal variable names in C
  - May only consist of letters, digits, and underscores
  - May be as long as you like, but only the first 31 characters are significant
  - May not begin with a number
  - May not be a C **reserved word (keyword)**

Reserved Words (Keywords) in C

- auto
- case
- const
- default
- double
- enum
- float
- goto

break
char
continue
do
else
extern
for
if

int
register
short
sizeof
struct
typedef
unsigned
volatile

long
return
signed
static
switch
union
void
while

Naming Conventions

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

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.

Legal Identifiers vs. Naming Conventions

- **Legal identifiers** refer to the restrictions C places on naming identifiers, i.e. variable names cannot begin with a number.
- **Naming conventions** refer to the standards typically followed by programmers, i.e. separating words with mixed case or underscores.

Which Are Legal Identifiers?

AREA

lucky***

Last-Chance

x_yt3

num$

area_under_the_curve

3D

num45

#values

pi

%done

Which follow the Naming Conventions?

Area

Last_Chance

x_yt3

finaltotal

area_under_the_curve

person1

values

pi

numChildren

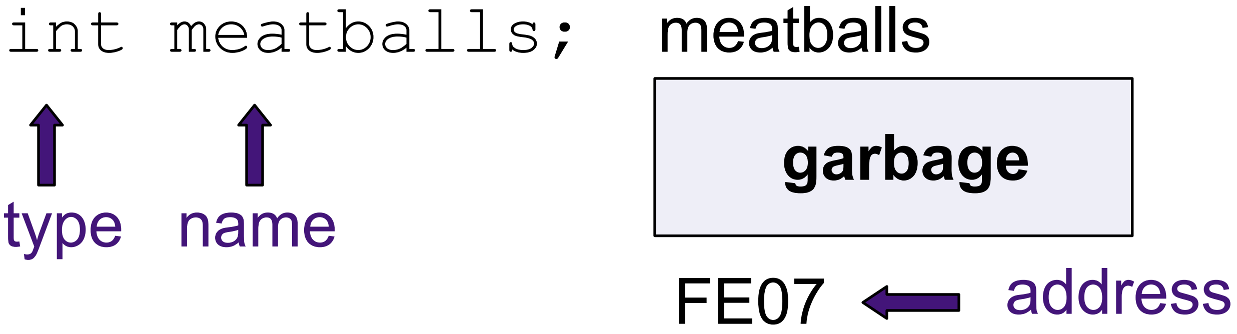
Declaring Variables

- 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;
```

More About Variables

C has three basic predefined data types:

- Integers (whole numbers)
  - **int**, long int, short int, unsigned int
- Floating point (real numbers)
  - **float**, **double**
- Characters
  - **char**
- At this point, you need only be concerned with the data types that are bolded.

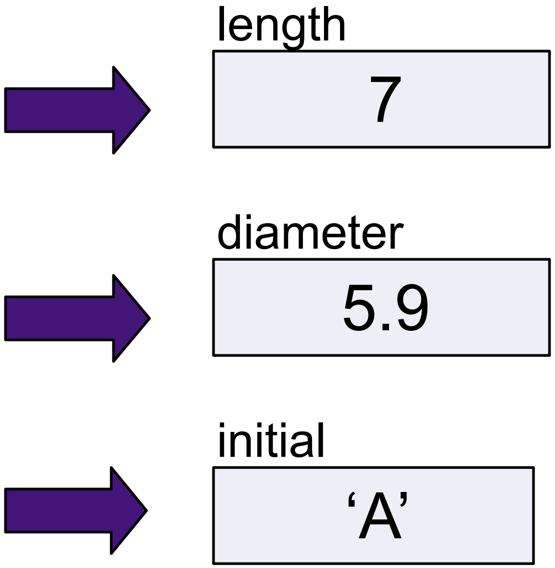
Using Variables: Initialization

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

```
int length = 7 ;

float diameter = 5.9 ;

char initial = ‘A’ ;
```



length

7

diameter

5.9

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   */
```

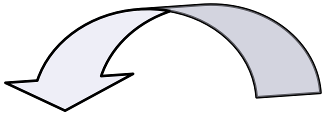
C will let you do the following:

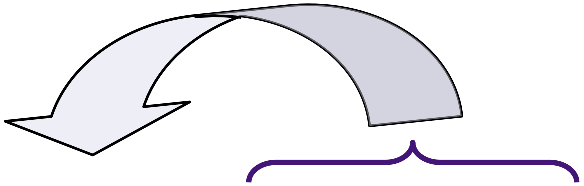
```
int height, width=6, area;
```

but it’s harder to read!

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 does not denote equality. It assigns the value of the righthand side of the statement (the **expression**) to the variable on the lefthand side.
- Examples:



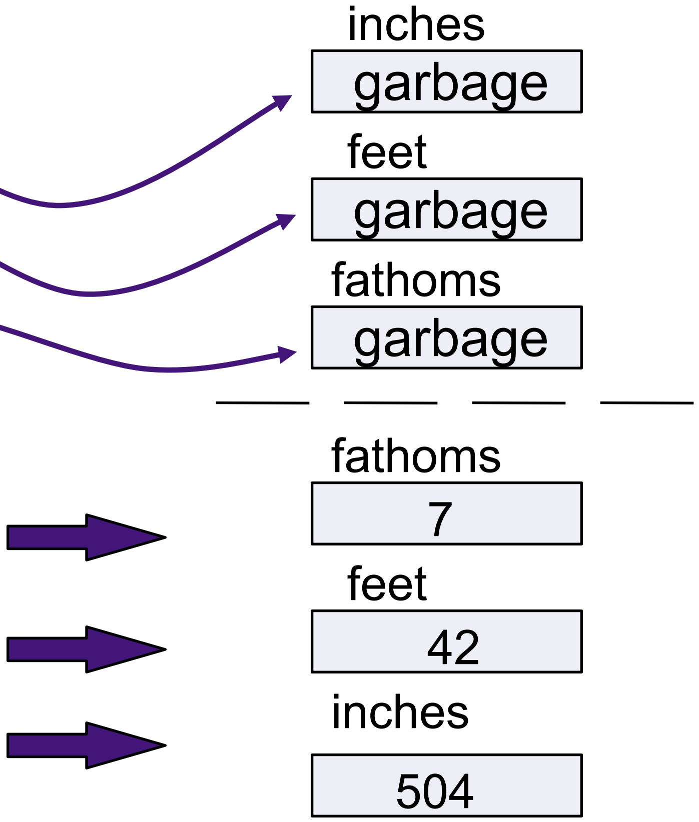


diameter = 5.9 ;   area = length * width ;

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.
- Floating point numbers can contain decimal portions.
- We can 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);
```

Final “Clean” Program (con’t)

```
    /* Convert the depth to inches */
    feet = 6 * fathoms;
    inches = 12 * feet;

    /* 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 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 spaces - be consistent!