CMSC 104 - Lecture 5
John Y. Park, adapted by C Grasso

Variables in C & Console I/O

Variables in C

Topics

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

What Are Variables ?

a + b

- **Variables** are <u>not</u> the same thing as variables in algebra.
  - A variable is the **name** the programmer assigns to a **location** in memory to store (remember) a value
  - The **value** of the variable is the contents of that memory location.

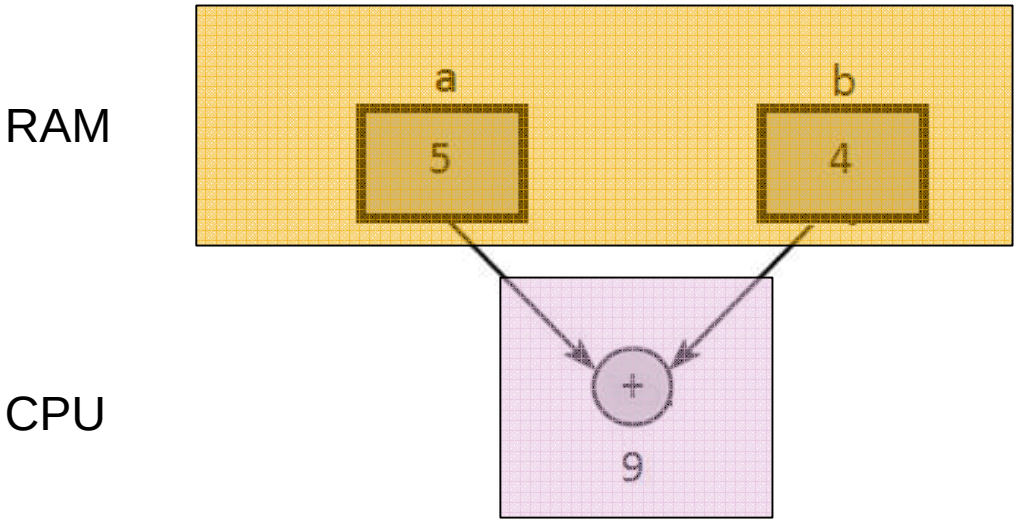
What Are Variables in C?

a + b

- When used in a C statement, the values that that exist in that location will be retrieved and loaded into the CPU
- Any operations that appear in that statement will be done using those values

What Are Variables in C?

a + b



Legal Identifiers in C

- We call the name for a variable in C an **identifier**
  - It identifies a location in memory
- Naming rules
  - May only consist of letters, digits, and underscores
  - May not begin with a number
    - Do not begin identifiers with underscores either
  - May be as long as you like
    - But it only looks at the first 31 characters
  - May not be a C **reserved word (keyword)**

Reserved Words (Keywords) in C

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

Case Sensitivity

- C is **case sensitive**
  - The case of each letter matters !
  - Example:

    area

    Area

    AREA

    ArEa

    are all seen as different identifiers by the compiler.

Legal Identifiers vs. Naming Conventions

- **Legal identifiers** refer to the restrictions **C** places on naming identifiers
  - Rules you have to follow
- **Naming conventions** refer to the standards you must follow for this course
  - Rules you should follow

CMSC 104 Naming Conventions

- **conventions** for naming variables.
  - Begin variable names with lowercase letters
  - Use meaningful identifiers (names)
    - Should describe what that variable is used for
  - Separate "words" within identifiers with underscores or mixed upper and lower case.
    - surfaceArea
    - surface_Area
    - surface_area
  - Be consistent!

Which Are Legal Identifiers?

AREA

lucky***

Last-Chance

x_yt3

num$

area_under_the_curve

3D

num45

#values

pi

%done

Which follow the CMSC Naming Conventions for variables ?

Area
Last_Chance
x_yt3
finaltotal
area_under_the_curve
person1
values
pi
numChildren

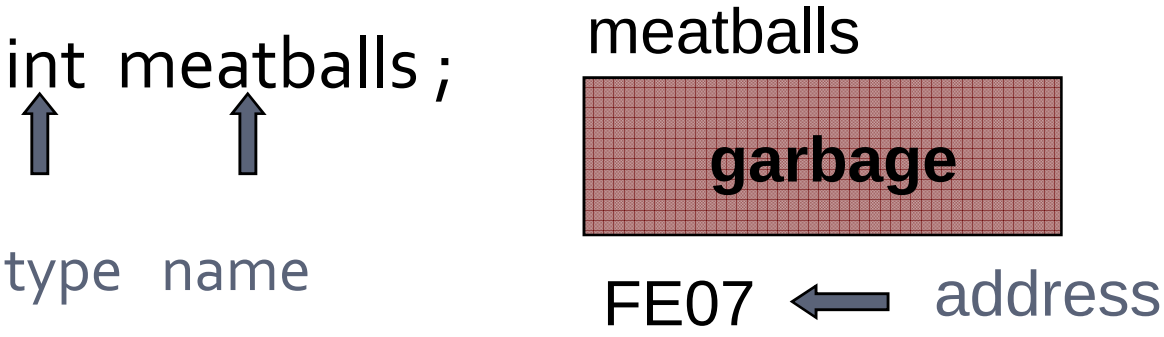
Declaring Variables

- Before using a variable, you must **declare** it.
  - gives the compiler some information about the variable and how much memory to set aside for it
- The **declaration statement** includes the **data type** of the variable.
  - Needs to know what kind of data it will hold.
- 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 has a unique **address**
- Visualization of the declaration



More About Variables

C has three basic predefined data types:

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

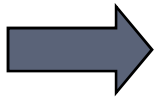
Variable Types

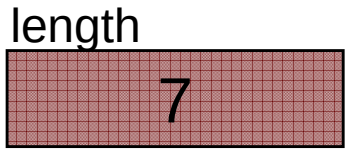
- int
  - Integer number that between -32767 and 32767
- float , double
  - Real number with an integral part and a fractional part
  - Can be written in C-style scientific notation
    - The letter 'e' or 'E' means "times 10 to the power"
    - 11,000 ⇔ 1.1e4
    - 0.00123 ⇔1.23e-3
- char
  - A single character (letter, number, special character)
  - Value must be wrapped by single quotes (' ')

Using Variables: Initialization

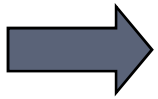
- Variables may be be given initial values, or **initialized**, when declared. Examples:

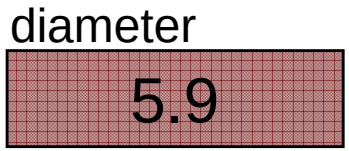
```
int length = 7 ;
```



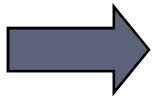


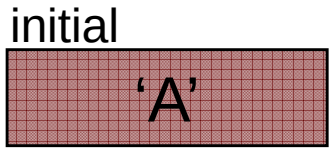

```
float diameter = 5.9 ;
```






```
char initial = 'A' ;
```





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 ;

The Assignment Operator

```
x = a + b
```

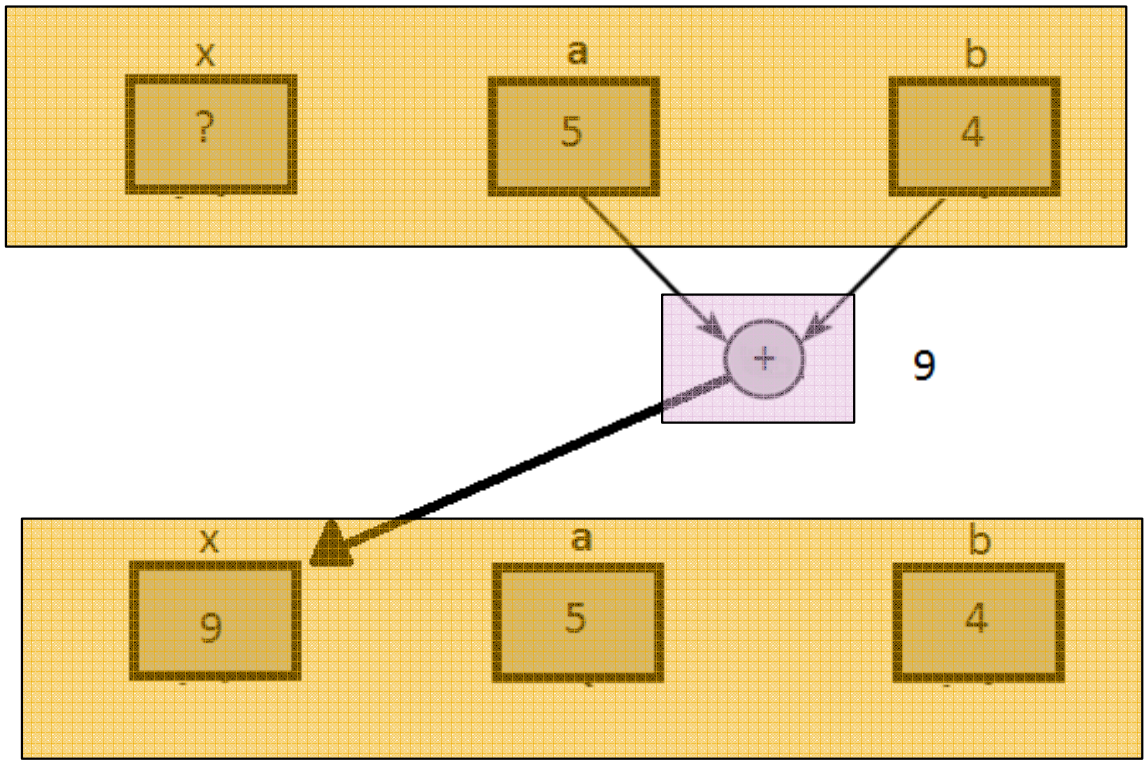
- The equal sign in C is <u>not</u> the same thing as equal sign in algebra
- **=** is the **assignment operator**
  - It computes the value of whatever is to the right of the **=**
  - Stores that value in the location named on the left hand side of the **=**
    - **There may only be <u>one</u> variable on the left side of the =**

The Assignment Operator

x = a + b

a + b

x =

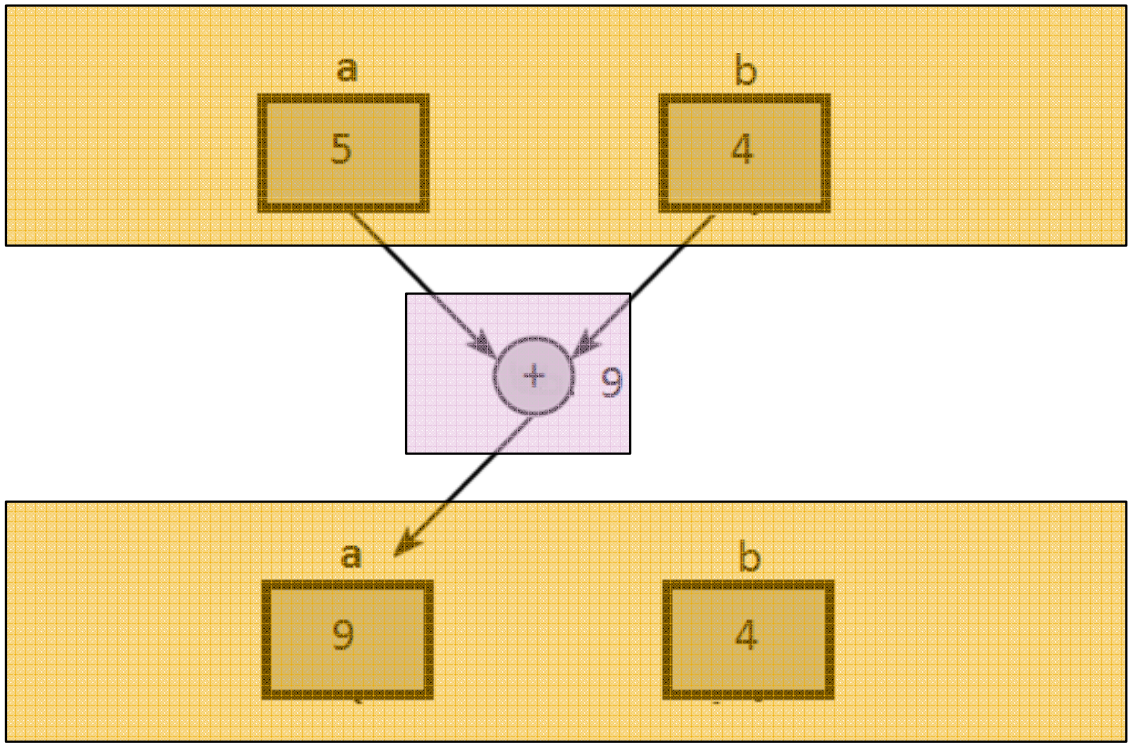


The Assignment Operator

a = a + b

a + b

a =



Data Input and Data Output

- Data Output
  - getting data from our application (in memory) out to the user or another application
  - **printf** will be used to print to the command line
- Data Input
  - getting data from outside our application into it
  - **scanf** will be used to read from the command line

Mad Libs

- mad libs
  1. Define some text with some blanks
     - Each blank specifies what kind of word to use
  2. Come up with words to fill the blanks

- Give me
  - 2 nouns – 1 syllable each
  - 2 verbs that rhyme- 2 syllables each

Mad Limerick

There once was a _____ from Id
Who stuck its _____ in a crib
It started to _____
And then it _____
And that was the last thing it did

printf

- Print text to the command line
- Example:
  - int num_children = 2;
  - printf("I have **%d** children", **num_children**);
- The %d is called a placeholder
  - It indicates that it is expecting an integer
  - It will be replaced in the string with the integer value that was passed to printf

printf, cont.

- Supported placeholders for printf
  - %c char
  - %d int
  - %f float
- Variable types must be compatible with placeholder types
- Can pass multiple placeholders to printf
  - printf("Their ages are **%d** and **%f** \n", 4, 1.5);

scanf

- Reads user input from the command line
- Parameters to scanf are:
  - A string containing placeholders
  - One or more pointers to variables
- Example
  - char firstInit;
  - printf("Enter first initial> ");
  - scanf("%c", &firstInit);
  - printf("You entered: %c \n", firstInit);

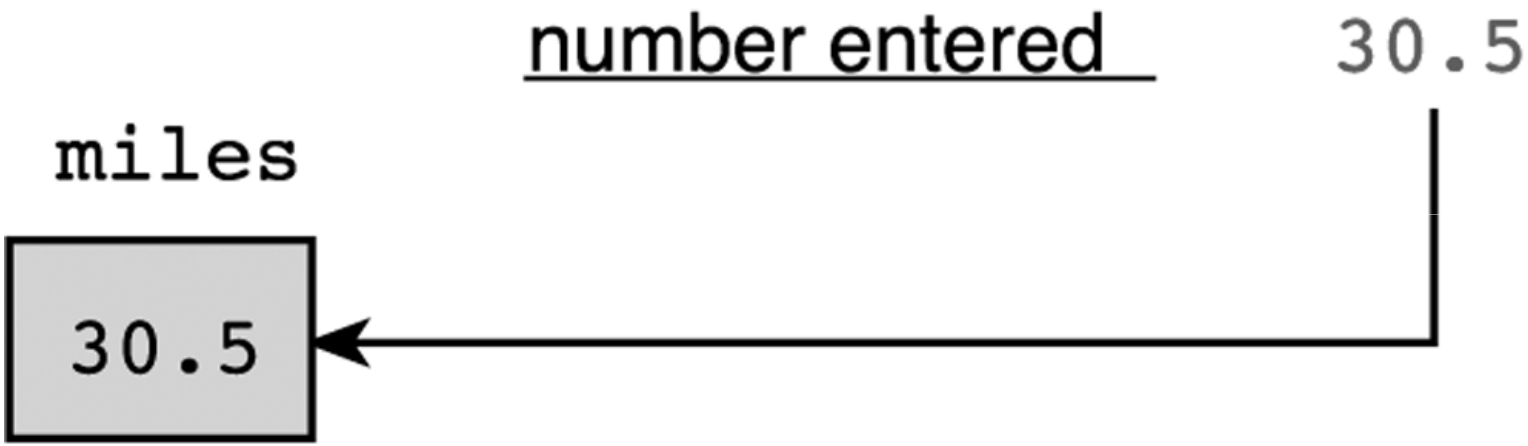
scanf, cont.

- You can prompt for more than one input at a time
- Example

```
char firstInit;
char middleInit;
printf("Enter 2 initials> ");
scanf("%c %c", &firstInit, &middleInit);
```

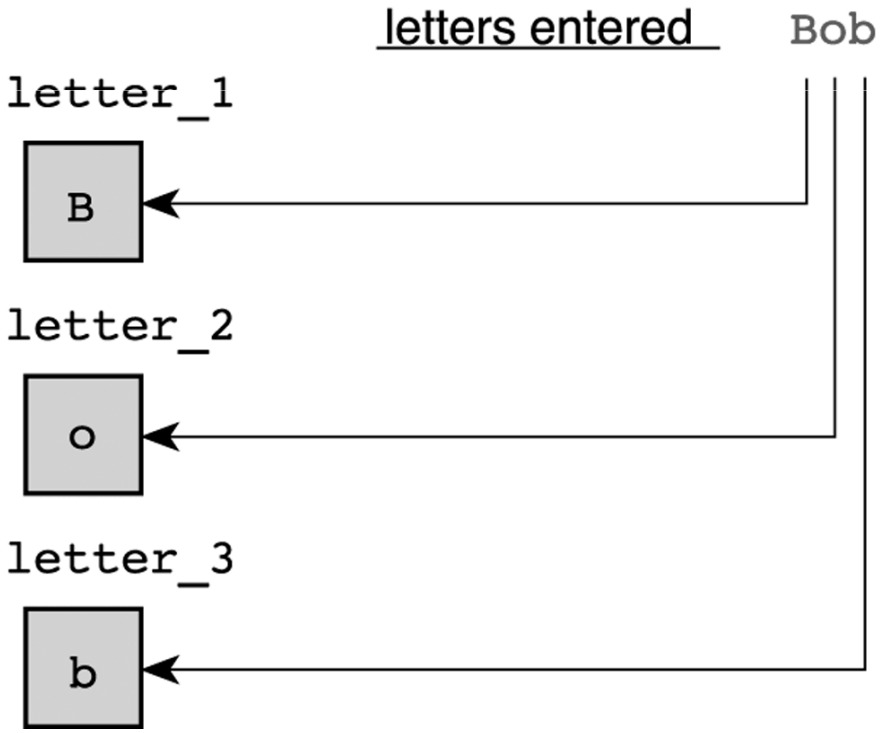
- When typed in, the values can be separated by space, newline, or nothing

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



Scanning Data Line "Bob"

```
char  c1, c2, c3;
scanf("%c %c %c", &c1, &c2, &c3);
```



Text values: char vs. string

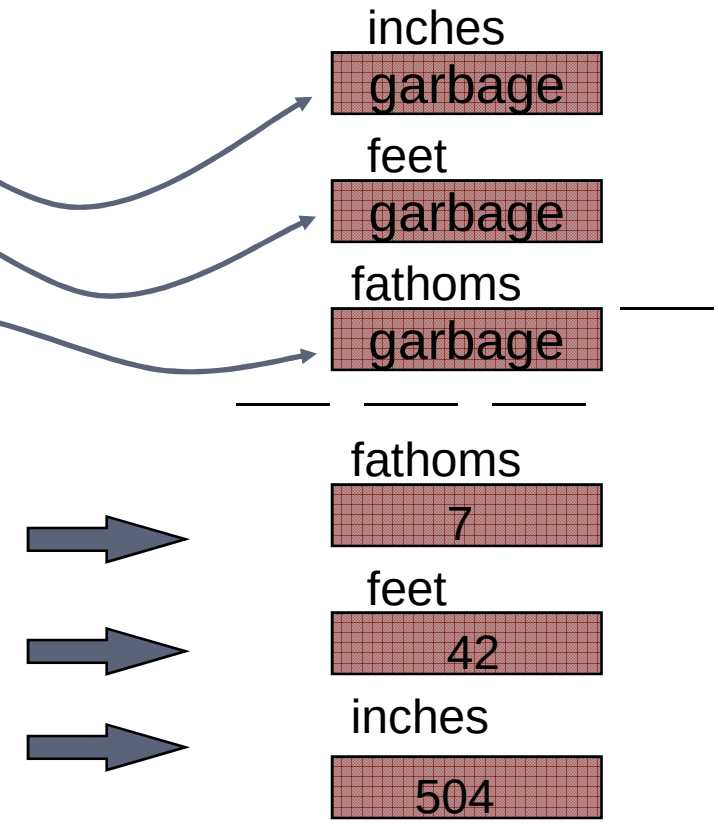
- chars
  - One and only one text character
  - Enclosed in **single quotes**
    - 'a'

- strings
  - One or more characters
  - Enclosed in **double quotes**
    - "I am a string"

Example: Declarations and Assignments

```
1.  #include <stdio.h>
2.  int main( )
3.  {
4.      int  inches, feet, fathoms ;

5.      fathoms = 7 ;
6.      feet = 6 * fathoms ;
7.      inches = 12 * feet ;
```



Example: Declarations and Assignments (cont'd)

```
8.      printf ("Its depth at sea: \n") ;
9.      printf ("    %d fathoms \n", fathoms) ;
10.     printf ("    %d feet \n", feet) ;
11.     printf ("    %d inches \n", inches) ;

12.     return 0 ;
13. }
```

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

```
1.  #include <stdio.h>
2.  int main ( ) {
3.      float  inches, feet, fathoms ;
4.      printf (“Enter the depth in fathoms : ”) ;
5.      scanf (“%f”, &fathoms) ;
6.      feet = 6 * fathoms ;
7.      inches = 12 * feet ;
8.      printf (“Its depth at sea: \n”) ;
9.      printf (“    %f fathoms \n”, fathoms) ;
10.     printf (“    %f feet \n”, feet) ;
11.     printf (“    %f inches \n”, inches) ;
12.     return 0 ;
13. }
```

NOTE: This program does not adhere to the CMSC104 coding standards

Final "Clean" Program

```
1.  #include <stdio.h>
2.
3.  int main( )
4.  {
5.      float  inches ;      /* number of inches deep   */
6.      float feet ;         /* number of feet deep     */
7.      float fathoms ;      /* number of fathoms deep */
8.
9.      /* Get the depth in fathoms from the user */
10.     printf ("Enter the depth in fathoms : ") ;
11.     scanf ("%f", &fathoms) ;
```

Final "Clean" Program (con't)

```
12.     /* Convert the depth to inches */
13.     feet = 6 * fathoms ;
14.     inches = 12 * feet ;
15.
16.     /* Display the results */
17.     printf ("Its depth at sea: \n") ;
18.     printf ("    %f fathoms \n", fathoms) ;
19.     printf ("    %f feet \n", feet);
20.     printf ("    %f inches \n", inches);
21.
22.     return 0 ;
23. }
```

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!