Variables in C

CMSC 104, Fall 2012
John Y. Park

Variables in C

Topics

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

What Are Variables in C?

- **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.

Legal Identifiers in C

- Another name for a variable in C is an identifier
- Variables in C may be given representations containing multiple characters. But there are rules for these representations.
- 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	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

CMSC104 Naming Conventions

- C programmers generally agree on the following **conventions** for naming variables.
  - Begin variable names with lowercase letters
  - 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 you must follow for this course, i.e. all variable names must begin with lowercase.

Which Are Legal Identifiers?

AREA	3D
lucky***	num45
Last-Chance	#values
x_yt3	pi
num$	%done
area_under_the_curve	

Which follow the CMSC104 Naming Conventions?

Area	person1
Last_Chance	values
x_yt3	pi
finaltotal	numChildren
area_under_the_curve	

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 ;

type name

meatballs

garbage

FE07 ← address

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 ; → length: 7

float diameter = 5.9 ; → diameter: 5.9

char initial = 'A' ; → 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:

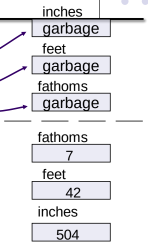
diameter = 5.9 ;   area = length * width ;

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

Example: Declarations and Assignments

```
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

```
#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 ;
 }
```

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

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 tab stops -- be consistent!