

Variables in C

Topics

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

Reading

- Sections 2.3 - 2.4

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.

Naming Variables

- Variables in C may be given representations containing multiple characters. But there are rules for these representations.

- Variable names (identifiers) 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 digit
- 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

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).
- Note: symbolic constants are not variables, but make the program easier to read.
- 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 different variables by the compiler.

Which Are Legal Identifiers?

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

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    int
```

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.

Notes About Variables

- You must not use a variable until you somehow give it a value.
- You can not assume that the variable will have a value before you give it one.
 - Some compilers do, others do not! This is the source of many errors that are difficult to find.



Using Variables: Initialization



- Variables may 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

- 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 does not denote equality. It assigns the value of the right-hand side of the statement (the **expression**) to the variable on the left-hand side.

- Examples:

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



Note that only single variables may appear on the left-hand side of the assignment operator.

Functions

- It is necessary for us to use some functions to write our first programs, but we are not going to explain functions in great detail at this time.
- *Functions* are parts of programs that perform a certain task and we have to give them some information so the function can do the task.
- We will show you how to use the functions as we go through the course and later on will show you how to create your own.

Displaying Variables

- Variables hold values that we occasionally want to show the person using the program.
- We have a function called `printf()` that will allow us to do that.
- The function `printf` needs two pieces of information to display things.
 - How to display it
 - What to display
- `printf("%f\n", diameter);`



`printf("%f\n", diameter);`

- The name of the function is "`printf`".
- Inside the parentheses are:
 - print specification, where we are going to display:
 - a floating point value ("`%f`")
 - We want to have the next thing started on a new line ("`\n`").
 - We want to display the contents of the variable `diameter`.
- `printf()` has many other capabilities.



Example: Declarations and Assignments

```
#include <stdio.h>
int main( void )
{
    int inches, feet, fathoms ;
    fathoms = 7 ;
    feet = 6 * fathoms ;
    inches = 12 * feet ;
}
```



Example: Declarations and Assignments

```
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 by using the `scanf()` function.



Enhanced Program

```
#include <stdio.h>
int main ( void )
{
    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 ;
}
```



scanf ("%f", &fathoms);



- The `scanf()` function also needs two items:
 - The input specification `"%f"`. (Never put a `"\n"` into the input specification.)
 - The address of where to store the information. (We can input more than one item at a time if we wish, as long as we specify it correctly.)
- Notice the `"&"` in front of the variable name. It says to use the address of the variable to hold the information that the user enters.

Note About Input and Output

- Whenever we wish to display values or get values from the user, we have a format problem.
- We can only input characters, not values.
- We can only display characters, not values.
- The computer stores values in numeric variables.
- `printf()` and `scanf()` will automatically convert things for us correctly.

Final “Clean” Program

```
#include <stdio.h>

#define FEET_PER_FATHOM 6
#define INCHES_PER_FOOT 12

int main( void )
{
    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

```
/* Convert the depth to inches */
feet = FEET_PER_FATHOM * fathoms ;
inches = INCHES_PER_FOOT * 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 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

- 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 5 spaces -- be consistent!
- Comments should explain why you are doing something, not what you are doing it.
a = a + 1 /* add one to a */ /* WRONG */
/* count new student */ /* RIGHT */

Another Sample Program

```
#include <stdio.h>
#define PI 3.14159
int main ( void )
{
    float radius = 3.0;
    float area;

    area = PI * radius * radius;
    printf( "The area is %f.\n", area );
    return 0 ;
}
```