Variables, Input and Output

Topics in this unit

- Variables as data containers
- Data types
- Input statements and input functions
- Performing Calculations
- Output statements

Structure of Code

- To recap, most programs you write will get input
- Process the input to produce some results
- Show output
- Input from the user is stored into variables
- Results are also stored into variables
- How do we create and use variables? Next…

Overview of C:
Variables and Data types

- To create a variable, it must be declared
- Declaring a variable involves choosing a name, and a data type
- The data type decides on what type of values can be saved and how much RAM is used

Datatype	**Description**
int, short, long	Whole numbers
double	Decimal numbers
string	Characters
char	A Single Character

Variable Names

- A variable name must
  - begin with a-z, A-Z, or _
  - followed by a-z, A-Z, 0-9 or _
- Names Are Case-Sensitive With A Length Of 31 Characters
  - Maybe More Than 31, Depending On The Compiler Used
- It is always good practice to initialize variable values when they are declared to some value

Names that can’t be used

- You can’t use these lowercase names. They are used by the compiler – we call them keywords
- Full List in section 38 in the book

```
break   case    char
const   default do
double  else    extern
float   for     if
int     long    return
```

Literals

- A fixed value that can not be further simplified
- Three basic types in C
  - numeric 45 3.14159 -70
  - character ‘a’ ‘7’ ‘*’
  - string “Hello World!”

Literals and Variables Compared

- A literal is a fixed value that never changes
- A variable is a container for literals
- Variable content changes using assignment statements
- A variable has a name, while a literal does not

Variable Declarations

- Every variable in C must be declared
  - MUST occur at the start of a main program or function
  - associates a name with a datatype
- Syntax: type_name variable_name;
- Examples:

```
int i;
double d;
```

Assignment Statements

- After declaring a variable, you normally put input values in it
- To store values into a variable, use the assignment statement
- Common form: variable = expression;
  - Causes expression to be evaluated and the result assigned as the new value of the variable
- Examples:

  ```
  i=5;
  i=i+1;
  y=m*x+b;
  ```

Assignment Statements

- Assignment statement is made of 3 components: Left side of =, = sign, and the right side.
- The left side of = must be a variable name (other entities are allowed but let’s skip that for now)
- The right side of = can be another variable, a literal, or an expression involving variables and literals.

Assignment Statement Example I

X=3;

Right side of = is a literal 3

Left side is a variable X

If X contains anything, it will be thrown out and 3 is saved into X.

Assignment Statement Example II

y=X+2;

X+2 is an expression. It is evaluated to a literal, then y is emptied out and the resulting literal of the expression is saved into y

If X is 3 (from previous slide) then right side evaluates to 5, then 5 is saved into y

Can’t Do’s

```
1=7;          ← left side is not a variable
5=x;          ← left side is a literal not a variable
“U”=“2”;      ← left side is a string literal
x+y = 45;     ← left side is an expression
```

Four Basic Types and Modifiers

- 4 basic types: int, char, float, and double
- Can extend to additional types using short, long and unsigned

What is a Constant

- A variables may change its content using assignment statements
- A constant is just that: it does not change its content
- Examples: pi is 22/7
- To create a constant use either: const or #define

Variable Scope

- Scope decides ‘where’ is a variable or a function can be used

Static Variables

- For now, we will not use static variables

- To simplify, using the static keyword makes a variable stay in memory after the function containing them is completed

Getting Input

- Many functions exist to get input
- Sometimes we get input but we don’t save it into variables, such as “press any key”. We get a key press from the user but we throw it out:

```
scanf_s(”%c”,&ch);
```

- More importantly, we get input, save it into a variable, then we process it.
  - Another version of scanf is used.

Input Statements

- Many functions exist based on what type of input and where it’s coming from
- Use scanf to get input from the Console
- scanf can be used to get input and save it to a variable
  - scanf(“%s”, &age); ← saves input into variable age

Input Statements II

- Another example: scanf(“%d”, &i);
  - reads from the Console into the variable
  - the quoted “%d” is a formatting code which instructs C how to interpret the input value
    - Use “%d” for storing integer values
    - Use “%f” for floating-point values
    - Use “%s” for strings
    - Use “%c” for a single character
  - In the above example, i must be declared int

Output Statements

- C output statement: printf(“%d”, i);
  - sends the value of variable i to the Console
  - the quoted “%d” is a formatting code which instructs C how to interpret the value to be printed
    - Use “%d” for integer values
    - Use “%f” for floating-point values
    - Use “%s” for strings
    - Use “%c” for a single character

Time For Our Next Demo!

```
/*
 Name: John Smith    Date: Jan 1st, 1900  -  This is called top level comments
Include top level comments in every assignment
final grade is the average of a midterm and  a final exam
 */

#include <stdio.h>

int main() {

/*
 C Programs must declare variables at the top  of a function which uses them.  After
the first   executable statement, you cannot declare any   more variables. You must
plan ahead and request the   variables that you need up front.
   It is always a good idea to initialize your  variables with a value to avoid the bug
caused by using a variable that has not be initialized  which happens a lot to new
programmers.
   C, unlike Visual Basic, will throw garbage values  into your uninitialized variables, so
lacking  initializations will always result in computations   that are not  correct.
 */
            int score = 0;
            double total = 0;
            double average = 0;
  /*
              It is always a good idea to output a prompt
              for the data you are trying to read.
   */

            printf( "I'm going to calculate averages\n" );
   printf( "based on a midterm and final score\n" );
   printf( "Gimme your midterm score:" );

/*
scanf function is how you read from the Console.  The first string is the formatting
string which C uses to figure out how to read the value.  %d is used to read int values.
Scanf stores the input into a variable that is always preceded by the & symbol. This
means we are reading a value directly  into the score variable.
*/

   scanf( "%d", &score );
   total = score;
   printf( "Gimme your final score:" );
   scanf( "%d", &score );

   total = total + score;
   average = total / 2;

/*
printf function accepts the same formatting codes used by scanf. %f is a placeholder
for the value in the variable listed at the end of  the call which is the average variable.
When working with printf, no & symbol can be used before the variable name.
Until we learn more, please try to remember this difference  between scanf and printf.
*/

   printf( "Your average is %f\n", average );

   return( 0 );

}
```

Summarizing Our Second Demo!

- To get input, show a prompt then get input.
- Variables hold a single value at a time·
- Arithmetic Operations allow us to calculate results: + - / *
- Special Characters: \t \n \" \' \? \\ %%
- Read all the comments carefully in the demo

Variable Initialization

- A variable has no meaningful value unless assigned
- Rule: Set each variable before its value is used! GARBAGE, otherwise! (The container has no predictable value)
- Some languages set variables to initial default values – there is no compiler set default in C
- One way of avoiding uninitialized variables: initialize at the time of declaration
  - `int your_sum = 20;`
  - `double rate=0.1 , balance=0.00;`

Time For Our Next Demo!

```
/*
   Let's try working with some other datatypes...
 */

#include <stdio.h>

int main()
{

 /*
 C provides many different datatypes.  Each one has a set of "valid values".  As a
programmer, you must ensure that you are always within this valid set of values.
*/

  short s = 12;   /* valid values: -32768 to 32767 */
  long l = 12;    /* valid values: -2147483648 to 2147483647 */
  char c = 'A';    /* valid values: one keyboard letter, but also 0-255 */
  unsigned int posValue = 12;  /* valid values: 0 - 65535 */
  unsigned long posBigValue = 12; /* valid values: 0 - 4294967295 */
  float f = 12.5;   /* a real value using single-precision */
  double d = 12.5;  /* a real value using double-precision */

/*
 You might wonder about what happens if we walk outside
 the set of "valid values" for a particular datatype.
  */

 printf( "A short is stored in your computer in %d bytes\n", sizeof( s ) );
 printf( "A long is stored in your computer in %d bytes\n", sizeof( l ) );
 printf( "A char is stored in your computer in %d bytes\n", sizeof( c ) );
 printf( "A unsigned int is stored in %d bytes\n",      sizeof( posValue ) );
printf( "A unsigned long is stored in %d bytes\n", sizeof( posBigValue ) );
printf( "A float is stored in your computer in %d bytes\n", sizeof( f ) );
printf( "A double is stored in your computer in %d bytes\n", sizeof( d ) );


   /*
 This next section shows you the formatting string for
  the different datatypes shown in this program.
    */

  printf( "Here is your short: %hd\n", s );
  printf( "Here is your long: %ld\n", l );
  printf( "Here is your char: %c\n", c );
  printf( "Here is your unsigned int: %u\n", posValue );
  printf( "Here is your unsigned long: %lu\n", posBigValue );
  printf( "Here is your float: %f\n", f );
  printf( "Here is your double: %lf\n", d );

   /*
          This next section shows you the formatting string for
          specifying a width to the values C prints out.
    */

        printf( "float: %5.2f double: %10.4lf\n", f, d );
        printf( "Notice the leading spaces before the value of the double!!\n" );

        return( 0 );
}
```

Summarizing Our Third Demo!

- Variables are typed memory locations
  - Datatype determines size in bytes
- When choosing a datatype, be mindful of the valid range of values
- Your compiler may yield different results when running our datatypes demo!

Summary

- Review each data type
- Variables vs constants
- Performing calculations
- Coding style