Lab 2: Input/Output and integer arithmetic

Goal

Read values into variables, perform arithmetic operations, and print the results.

Background

Variables

- A *variable* is symbolic name for a memory location. It may be used within a program to save a value that may be retrieved and used at some later time during the execution of the program.
- The C language supports two distinct primitive classes of variables: *integer* and *floating point* (which is an approximation of the real number system)
- Our focus today will be on the *integer class* of which there are (at least) four specific *types*: char, short, int, and long.
- An integer variable is declared via a statement of the following form:

  optional_modifier type_name variable_name;

- The default *modifier* is *signed*. To create an *unsigned* integer the *unsigned* modifier must be explicitly provided. The number of bits associated with each type is shown below.

  char - 8 bit
  short - 16 bit
  int - 32 bit (sometimes 16)
  long - 32 bit (sometimes 64)

Example declarations:

```
unsigned char    value1;
int              xyz;
unsigned short   has16bits;
long             this_is_the_sum;
```

Variable names:

- Comprised of upper and lower case letters, digits, and _
- Must start with a letter or _
- *Must not be* a *reserved word (e.g. int, void, while, char, unsigned ....) See appendix A of text*
- *Should not be* the name of a standard function or variable.... but *how do I know what those are?*

Performing operations on variables:

- ***Operators*** **can be grouped into several classes.**

Assignment: = Assigns the value of the expression on the right side to the *variable* on the left side.

Arithmetic: +, -, *, /
Add, subtract, multiply, divide

Comparative: ==, != , <, <=, >, >=
equal, not equal, less than, less than or equal to, greater that, greater than or equal to

Expressions are built by combining operators and operands

- Suppose the variable *v1* currently has value *4* and *v2* has value 3.
- What is the value of the following expression?
  - v1 + 5 * v2 / 3 * v1
- The C compiler has a set of immutable rules for evaluating such expressions. The rules are called *precedence and associativity rules* but they may be hard to remember.
- We can ensure that the compiler does what we want by building our expressions from parenthesized sub-expressions. Such expressions are *always* evaluated *innermost parentheses first*. All of these expressions may have different values:

```
v1 + ((5 * v2) / (3 * v1))

v1 + (5 * (v2 / 3)) * v1

((v1 + 5) * v2) / (3 * v1)
```

Introduction to Input and Output

Useful computations usually require a mechanism by which the user of a program may provide *input data* that is to be assigned to variables, and another mechanism by which the program may produce *output data* that may be viewed by the user.

- Initially, we will assume that the user of the program
  - enters input data via the keyboard and
  - views output data in a terminal window on the screen.
- The C run time system automatically opens two files for you at the time your program is started:
  - *stdin* – Input from the keyboard
  - *stdout* – Output to the terminal window in which the program was started
- When your program uses input/output or other functions in the C language run time library, *you must include proper header files.* These header files:
  - contain the declaration of variables you may need
  - allow the compiler to determine if the values you pass to the function are correct in type and number.
- If your program is to use the standard library functions for input and output, you must include the header file *stdio.h* as shown below.

```
#include <stdio.h>

int main()
{
```

Input with the fscanf() function

- The fscanf() function reads input data from the standard input which is normally bound to the keyboard of the computer on which the program is running.
- Example:

```
int v1, v2;  // variables to hold input values
int howmany; // number of values successfully read

howmany = fscanf(stdin, "%d %d", &v1, &v2);
```

- The '%d' specifies that the variable to be read must be an integer, the '&v1' and '&v2' specifies that the value should be placed into the memory locations occupied by the v1 and v2 variables. The & (address of) operator is mandatory because fscanf() must be told *where* to store the value it retrieves.
- The fscanf() function returns the number of values it acquired and this value will be saved in the *howmany* variable. If the user of the program enters two integer values such as:

  15 63

  howmany will be assigned a value of 2. But if the user should enter

  14 abcd

  then fscanf() will abort when it discovers abcd is not a valid integer and *howmany* will have the value 1. In this case the value of v2 will be unchanged (and possibly unknown).

Output with the fprintf() function

The fprintf() function generates output data to the standard output which is normally bound to the display of the computer on which the program is running.

- Example:

```
int v1, v2;  // variables holding values to be printed
int howmany; // number of values successfully printed

v1 = 99;
v2 = 1234;
howmany = fprintf(stdout, "The values are \n");
howmany = fprintf(stdout, “v1 = %d and v2 %d \n”, v1, v2);
```

- The '%d' specifies the variable to be printed is be an integer, the 'v1' and 'v2' specifies that the values of the variables v1, and v2 are to be passed to fprintf(). The & (address of) operator *must not be used here* because fprintf() must be passed the values to be printed and not their addresses.
- Literal text such as the 'v1 =' may be interspersed with output values as shown above.
- The \n character automagically starts a new line of output.
- Analogous fscanf(), the fprintf() function returns the number of values it printed.

Assignment:

Write a complete C language program named lab2.c that will

- use fscanf to read two integer values into variables v1 and v2
- assign the value of each of the following expressions to variables e1, e2, and e3.

```
v1 + ((5 * v2) / (3 * v1))

v1 + (5 * (v2 / 3)) * v1

((v1 + 5) * v2) / (3 * v1)
```

- and then print the values of e1, e2, and e3 in the following form

v1 = 3 and v2 = 8
Expression values are:
e1 = 7, e2 = 33, and e3 = 7

Redirection of input and output

When a program is run in the Unix environment, the logical file *stdout* is by default associated with the screen being viewed by the person who started the program.

The > operator may be used on the command line to cause the standard output to be *redirected* to a file:

```
./a.out  >  output.txt
cat output.txt
```

```
v1 = 3 and v2 = 8
Expression values are:
e1 = 7, e2 = 33, and e3 = 7
```

Like the *stdout* the *stdin* may also be redirected. To redirect both *stdin* and *stdout*, use your favorite editor to create a file called input.txt containing two integer values and then do:

```
./a.out  < input.txt >  output.txt
```

Turn In Work

Show your TA that you completed the assignment. Then turn in your lab2.c program using the command:

sendlab.101.*section_number* 2 lab2.c