CpSc 1111 Lab 4
Integer Variables, Mathematical Operations, ASCII, & Redirection

Overview

By the end of the lab, you will be able to:

- declare variables
- perform basic arithmetic operations on integer variables
- use `fprintf` to print out integer and character values
- use redirection to send output to a file
- use the Unix command `cat` to echo the contents of a file to the screen

Background Information

If you remember from lab02, you were given a program with errors that you had to correct. The program contained variables as well as `printf()` and `scanf()` function calls. This week, you will be writing a program declaring variables, initializing some and assigning results of mathematical expressions to others. You will see some of the precedence rules with the mathematical expressions. Some background information of the concepts from this lab are shown below, some repeated from previous weeks and some new.

Variables

As you learned previous weeks in lab and lecture, a variable is a symbolic name for a memory location where values are stored. We, as programmers, can access the memory locations needed in our programs much easier by name rather than by the actual address of the memory location.

In the C language, every variable must be declared. To declare a variable, its type must be indicated: `int`, `float`, `bool`, `double`, `char`, etc. Note that for `bool` you need to `#include <stdbool>`. It is preferable to use meaningful variable names, which improves program readability.

Integer Variables

You also learned previous weeks that the `int` type is used for integral numbers, or integers – numbers without decimal places. Don't forget that usually, variables are all declared at the beginning of a function. The following are examples of `int` variable declarations:

```
int sideLength;
int area;
```

Variables that have not been given a value at any point contain garbage – they will have the value of whatever had been in that memory location the last time it was used. In C, you must never assume that variables have an initial value of 0 upon declaration.

You can ***initialize*** a variable upon declaration (such as `sideLength` below), or ***assign*** it a value at some point later in the program (`area` below).

```
int main(void) {
      int sideLength = 3;
      int area;

      area = sideLength * sideLength;
      printf("The area of a square with side length of %i ", sideLength);
      printf("is:  %i \n", area);

      return 0;
```

```
}
```

There are also optional ***modifiers*** that can be used with integers: `long`, `long long`, `short`, `unsigned`, and `signed`. An integer, by default, is a signed value. For an integer to be a signed value, you can declare it as `signed int` or just simply as `int`.

The format for declaring an integer variable is the following:

<optional modifier> <type_name> <variable_name>

The number of bits that a data type uses depends on the system. For example, integers are guaranteed to be at least 32 bits big, but sometimes may be 64 bits big. On our system, they take up 32 bits.

Type	Size
`short`	16 bits (guaranteed to be at least 16 bits)
`int`	32 bits (guaranteed to be at least 32 bits; sometimes they are 64 bits)
`long int`	64 bits
`long long int`	64 bits

The following are examples of different integer variable declarations:

```
int             value1;          // signed value by default
                                 // at 32 bits big, the range is from:
                                 //     -2,147,483,648 to +2,147,482,647

unsigned int    value2;          // at 32 bits big, the range is from:
                                 //     0 to +4,294,967,295

unsigned short has16bits;        // guaranteed to be at least 16 bits
                                 // (it is 16 bits on our system)

long int        the_sum;         // 64 bits
```

Basic math operations can be performed on integer variables. The following table shows a few examples.

Example	Description
`y = 3;` `x = 5;`	assigns the value 3 to y and 5 to x
`z = x + y;`	assigns the sum of x and y to z
`z = x + 5;`	assigns the sum of x and 5 to z
`x += 4;`	same as `x = x + 4;` adds 4 to x
`a = x - y;`	assigns the difference of x and y to a
`z -= 1;`	same as `z = z - 1;` subtracts 1 from z
`z *= 2;`	same as `z = z * 2;` multiplies z by 2
`z = x * y;`	assigns the product of x and y to z
`z = x / y;`	assigns the integer dividend of x and y to z *** Be CAREFUL with integer division!**

Characters and ASCII Code

In the C programming language, characters are a special type of integer. Each character is represented by an 8-bit binary code, shown on the ASCII table. This ASCII table can be used as reference. For example, you can see from the table that the capital letter A has a integer (decimal) value of 65, which means that it is represented by the binary value 01000001.

The following `printf()` statements may help to understand the character type and their relationship to ASCII:

```
printf("%c", 'A');      would print to the screen:  A
printf("%d", 'A');      would print to the screen:  65
printf("%c", 65);       would print to the screen:  A
```

Performing Operations on Variables

Expressions consist of a collection of operators, variables, and constants.

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 to, not equal to, less than, less than or equal to, greater than, greater than or equal to

Statements are expressions followed by a semicolon. The most commonly encountered statements are:

Assignment statement:

```
x = y + 2;
```

Function invocation:

```
howmany = scanf("%d", &v1);
howmany = fscanf(stdin, "%d", &v1);   // fscanf() discussed further down
printf("Hello World!\n");
```

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*** (e.g. multiply and divide are done before add and subtract) and ***associativity*** (e.g. a collection of multiply and divides is evaluated left to right so `42 / 6 * 7 = 49` and NOT `1`), but they are 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 have different values.
  What values will each of these expressions have?

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

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

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

---

fprintf() and fscanf()

From lab02, the lab write-up contained some background information about the 3 files that the C run-time system automatically opens up when a program is run. **Do you remember what they are?**

You already know that the `printf()` function prints to standard output (usually the screen), and the `scanf()` function gets input from standard input (usually the keyboard).

The `fprintf()` and `fscanf()` functions can be used as well. These are described below.

fprintf()

The `fprintf()` function is similar to the `printf()` function except that `fprintf()` has an additional argument as the first argument which specifies where the output will be sent to. If sending output to the screen, the following two statements are essentially the same in that they will both send the same message to the terminal window:

```
printf("I love programming!\n");
fprintf(stdout, "I love programming!\n");
```

But, later in the semester, you will see that you can send output to a file specified as the first argument in the `fprintf()` function.

fscanf()

The `fscanf()` function is similar to the `scanf()` function except that `fscanf()` has an additional argument as the first argument which specifies where the input is coming from. If the input is coming from the keyboard, the following two statements are essentially the same in that they will both be getting the input from the keyboard and putting it into the memory location (variable) that we are calling `userInput`:

```
scanf("%d", &userInput);
fscanf(stdin, "%d", &userInput);
```

But, later in the semester, you will see that you can get input from a file, specified as the first argument in the `fscanf()` function.

More Fun With Unix

In Unix, you can use a ***redirection*** operator to send output that would otherwise be going to the screen to a file. You can also use redirection for input as well. For this lab, you will use redirection for the output.

For example, when executing a 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
```

Unix also provides a command that will echo the contents of a file to the screen. To view the contents of a file, you could use your editor of choice to open the file, or you can use the following Unix command, which will show the contents to the screen without opening the file in the editor:

```
cat output.txt
```

Lab Assignment

Create a file called `main.c`. In it, provide a C program that will do the following:

1. Declare four integers variables `intVar1`, `intVar2`, `intVar3`, and `intVar4`.

2. Initialize `intVar1` to the value `4`; initialize `intVar2` to the value `5`.

3. Declare four more integer variables `exp1`, `exp2`, `exp3`, and `exp4`.

4. Declare a character variable `charVar`.

5. Assign the value of each of the following expressions to variables `exp1` and `exp2` respectively:
```
exp1 = intVar1 + ( (5 * intVar2) / (3 * intVar1) );
exp2 = intVar1 + (5 * (intVar2 / 3)) * intVar1;
```

6. Using the `fprintf()` function, print to standard output the values of the variables (your output should look like the following):
```
intVar1 = 4  and intVar2 = 5

Expression values are:
exp1 = 6
exp2 = 24
```

7. Compile and run and verify that your output looks like the above lines. Notice that using the `-Wall` option with `gcc` will produce warnings about unused variables at this point. You should always fix any warnings that you get when compiling because many times, they are indicative of a problem that does affect your program.

8. Assign to `intVar3` the value `9`; assign to `intVar4` the value `5`.

9. Assign the value of each of the following expressions to variables `exp3` and `exp4` respectively:
```
exp3 = (intVar4 % 2) / (intVar4 / intVar3);
exp4 = 2 + intVar3 * intVar4;
```

10. Using the `fprintf()` function, print to standard output the values of the variables from step #8. Compile and run. What happens and why? Do you get the expected new output? **Write your answer in the header comment portion of your `main.c` file.**

11. Change the value of `intVar3` to `5` and change the value of `intVar4` to `9`. Then compile and run and verify that your output looks like this:
```
intVar1 = 4  and intVar2 = 5

Expression values are:
exp1 = 6
exp2 = 24

intVar3 = 5  and intVar4 = 9

Expression values are:
exp3 = 1
exp4 = 47
```

12. Assign to `charVar` the value `'H'`.

13. Using the `fprintf()` function, print `charVar` as a character and the ASCII value of `charVar` (using the appropriate format specifiers to display `charVar` as a character and as an integer respectively)

14. Increase the value of `charVar` by 32:

```
charVar += 32;
```

15. Use an identical `fprintf()` statement as used in step 13 to print `charVar` as a character and the ASCII value of `charVar`. Then compile and run and verify that your output looks like this:

```
intVar1 = 4  and intVar2 = 5

Expression values are:
exp1 = 6
exp2 = 24

intVar3 = 5  and intVar4 = 9

Expression values are:
exp3 = 1
exp4 = 47

charVar = H ASCII value: 72

charVar = h ASCII value: 104
```

16. Run the program again but redirect the output to a file. The `>` operator may be used on the command line to cause the standard output to be ***redirected*** to a file:

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

Type `ls` to ensure that you do have an `output.txt` file. To view the contents of the text file that contains the output (called `output.txt` above), you can use your editor of choice to open the file, or you can use the `cat` command to show the contents of the file to the screen without opening the file in the editor. Use the `clear` command first and then the `cat` command:

```
clear
cat output.txt
```

You should see the following output printed to the screen (which should be the contents of your output file):

```
intVar1 = 4  and intVar2 = 5

Expression values are:
exp1 = 6
exp2 = 24

intVar3 = 5  and intVar4 = 9

Expression values are:
exp3 = 1
exp4 = 47

charVar = H ASCII value: 72

charVar = h ASCII value: 104
```

Turn In Work

1. Before turning in your assignment, make sure you have followed all of the instructions stated in this assignment and any additional instructions given by your lab instructor(s). Always test, test, and retest that your program compiles and runs successfully on our Unix machines before submitting it.
2. Show your TA that you completed the assignment. Then submit your `main.c` program and your `output.txt` file using the handin page: http://handin.cs.clemson.edu. ***Don't forget to always check on the handin page that your submission worked. You can go to your bucket to see what is there.***

Reminder About Style, Formatting, and Commenting Requirements

- The top of your file should have a header comment, which should contain:
  - Your name
  - Date
  - Lab section
  - Lab number
  - Brief description about what the program does
  - Any other helpful information that you think would be good to have.
- Variables should be declared at the top of the main function, and should have meaningful names.
- Always indent your code in a readable way. Some formatting examples may be found here: https://people.cs.clemson.edu/~chochri/Assignments/Formatting_Examples.pdf

Grading Rubric

If your program does not compile on our Unix machines or your assignment was not submitted on time, then you will receive a grade of zero for this assignment. Otherwise, points for this lab assignment will be earned based on the following criteria:

Functionality	75	
Style, Formatting, & Commenting	10	
Inclusion of correct output.txt	5	
Inclusion of answer in header comment	5	
Use of `fprintf()` function	5	
Other	-5	if warnings when compile, and other point deductions decided by grader