CSCE 155 - C

Lab 02 - Data Types

Dr. Chris Bourke

Prior to Lab

Before attending this lab:

1. Read and familiarize yourself with this handout.
2. Read sections Preliminaries, Basics of Compilation, Programming Structure and Style, and Variables from the C Wikibook (http://en.wikibooks.org/wiki/C_Programming) *or*
3. Read the following tutorial:
   http://www.codingunit.com/c-tutorial-variables-and-constants

Peer Programming Pair-Up

To encourage collaboration and a team environment, labs will be structured in a *pair programming* setup. At the start of each lab, you will be randomly paired up with another student (conflicts such as absences will be dealt with by the lab instructor). One of you will be designated the *driver* and the other the *navigator*.

The navigator will be responsible for reading the instructions and telling the driver what to do next. The driver will be in charge of the keyboard and workstation. Both driver and navigator are responsible for suggesting fixes and solutions together. Neither the navigator nor the driver is "in charge." Beyond your immediate pairing, you are encouraged to help and interact and with other pairs in the lab.

Each week you should alternate: if you were a driver last week, be a navigator next, etc. Resolve any issues (you were both drivers last week) within your pair. Ask the lab instructor to resolve issues only when you cannot come to a consensus.

Because of the peer programming setup of labs, it is absolutely essential that you complete any pre-lab activities and familiarize yourself with the handouts prior to coming to lab. Failure to do so will negatively impact your ability to collaborate and work with others which may mean that you will not be able to complete the lab.

1 Lab Objectives & Topics

At the end of this lab you should be familiar with the following

- Using command line arguments
- Variable declarations
- Basic primitive data types
- How to choose appropriate data types for a given problem

2 Activities

2.1 Using Command Line Arguments

When you run a program from the command line, you can also provide the program with *arguments* that the program can use for input or for configuration. To understand this, we have provided you two completed C programs that compute an age given a birthdate. The first works by prompting the user for input interactively, while the second uses command line arguments.

Instructions

1. Go to your `labs` directory and clone the Lab 02 project from Github by executing the following command:
   `git clone https://github.com/cbourke/CSCE155-C-Lab02`
2. Change your directory to the cloned project and open these files in your text editor of choice and examine them, but do not make any changes.
3. Compile the programs into executable called `birthday01` and `birthday02` respectively:
   ```
   gcc -o birthday01 birthday.c
   gcc -o birthday02 birthday_cli.c
   ```
4. Run `birthday01` and answer the questions on your worksheet

5. The second program is non-interactive: it reads input directly from the command line when you run it, so run it as:
   `./birthday02 Dennis 1941 9 9`
6. Run the program and answer the questions on your worksheet

2.2 Basic Data Types

A variable is a name associated with a memory cell whose value can change. When variables are stored in memory, the computer has to have a way of knowing what type of data is stored in a given variable. Data types identify the type of values stored in a memory location and operations that can be performed on those values. A computer will not use the same amount of memory to store a single letter as it does to store a very large real number, and the values are not going to be interpreted the same way. Therefore, different data types may have different sizes. The (incomplete) table in question 4 on the worksheet shows some basic data types with their sizes and ranges. Size is represented in bytes. A byte is a unit of storage capable of holding a single character. A byte is usually considered equal to 8 bits. A bit (short for binary digit), is the smallest unit of information in a computer–either a zero or a one. Some basic data types can be either signed (either positive or negative) or unsigned (non-negative).

Instructions

The sizes and ranges of data types depend on the system the program is compiled for. We will explore the sizes of each of the primitive types defined in C on the CSE system. You have been provided a source file, `ranges.c` that outputs the sizes of each of the basic types.

1. Compile the source file:
   `gcc -o ranges ranges.c`
2. Run your program: `./ranges`
3. Complete the table provided in your worksheet

2.3 Currency Conversion

Write a program that will convert US Dollars to British Pounds and Japanese JPY. 10% of the total amount of US Dollars will be taken as an exchange fee. For the rest of the US Dollars, half will be changed to British Pounds and the other half to JPY. Assume the exchange rate is: 1 US Dollar = 0.6 British Pound; 1 US Dollar = 76.8 JPY. The program should ask the user to input the amount of US dollars then print an appropriate output. An example run would look something like the following:

```
Please input the total amount of US Dollars: 100.00
You get 27.00 British Pounds and 3456.00 Japanese JPY.
```

Instructions

1. Using an editor of your choice, create your source file called `dollar.c`.
2. Write your complete C program in this source file, be sure to choose appropriate data types for your variables
3. Compile and run your program as before:

   ```
   gcc -o dollar dollar.c
   ./dollar
   ```

4. Test your program and answer the questions on your worksheet

2.4 Mixed types

A mixed-type expression is an expression with operands of different data types. When an assignment statement is executed, the expression on the right-hand-side is evaluated, and then the resulting value is placed into the variable on the left-hand-side. For example:

```
1 int x = (8 * 4) + (3 * .5);
```

The data types of the operand affect the data type of the result. This can lead to some initially unintuitive results. When performing division between two integers, the result is necessarily an integer. For example:

```
1 int a = 10, b = 20, c;
2 c = a / b;
```

In the code snippet above, the floating point result of `10 / 20` *should* be `0.5`, but the actual value stored in the variable `c` is zero! This is because the result of an operation of two integers is an integer: thus the decimal part of the result is truncated (dropped). We could fix this by making at least one of the operands (and the resulting variable) a floating point variable through type-casting:

```
1 int a = 10, b = 20;
2 double c;
3 c = (double) a / b;
```

You have been given a C program, `area.c` that reads in the base and the height of a triangle and calculates the total area.

1. Read through and understand the source code
2. Compile and run your program to answer the remaining questions on your work-

sheet.

3. Using the previous birthday programs as a reference, change the area program to accept command line arguments instead of prompting for input.

3 Handin/Grader Instructions

1. Hand in your `area.c` source file by pointing your browser to: https://cse-apps.unl.edu/handin and login with your CSE login/password.
2. Grade yourself by pointing your browser to https://cse.unl.edu/~cse155e/grade/ (you may need to change the course in this URL depending on which course you are taking)
3. Enter your cse login and password, select the appropriate assignment for this lab and click grade me.
4. You will be displayed with both expected output and your program's output. The formatting may differ slightly and that is not important. As long as your program successfully compiles, runs and outputs the same values, it is considered correct.

Turn your worksheet in to the lab instructor.

4 Advanced Activities (Optional)

1. Modify the `birthday02` program to *validate* the input data. If the data is invalid, force the program to quit with an error message.
2. Modify the `area.c` program to accept inputs from command line arguments rather than prompting the user for input. Command line inputs are communicated through the `**argv` argument to the `main` function (which is an array of strings). Each argument can be accessed by indexing this array with zero being the first index (so the arguments are stored in `argv[0]`, `argv[1]`, etc.). The first argument is always the executable's file name, so the first actual argument starts at index 1. To convert these strings to numeric values you can use the conversion functions in the standard library:

```
1 int a = atoi(argv[1]);
2 double b = atof("3.14");
```