Laboratory 3: Data Types and Operations

Lecture notes:

1. A quick review of lecture 2:
   a. Comment: /* */
   b. Variables: declaration, type, name, initialization, assigning value, constant
      In C, all variables must be declared before they are used.
   c. Input/output: printf(), scanf()
2. Sizes of data types on GLUE (may vary on other machines)

char	8 bits, one character
short int	16 bits, one integer from -32768 to 32767. (also known as *short)*
int	*32* bits, one integer (on GLUE, this is the same as *long int)*
float	*32* bits, single-precision (about 6.5 decimal digits) floating point
double	64 bits, double-precision (16 decimal digits) floating point
long double	128 bits, floating point with about 34 digits of precision on Glue; or 96 bits (12 bytes) to store Intel 80 bit format of about 25 digits of precision.

   sizeof(): returns the size of the parameter (in bytes)
   scientific notation for real numbers: -1.2e34 ($-1.2x10^{34}$), 2.5e-18 ($2.5x10^{-18}$)
3. Data type char:
   a. Represents a character as a 8-bit number (one byte)
   b. ASCII code: 0-255 decimal. (0: 48, A: 65, B: 66, ..., Z: 90, a: 97, b: 98, ...)
   c. Assign value to a char type variable

```
char grade = 'D';     /*declare grade and assign initial value*/
grade = 'C';            /*grade now changes to C*/
grade = grade - 1;    /*grade is upgraded to B*/
```

4. Array and string (basics):
   a. Declare an array/string: *type array_name[size_of_array];*
   b. Access array element: exactly the way of accessing/using a variable of the defined type and name *array_name[element_index]*
   c. Note: array index is 0-offset.
   d. String is an array of char type.
      i. \0 is the end of string character.
      ii. %s is the format indicator to read or print a string as a whole (rather than individual elements) by *scanf()* and *printf().*
   e. Examples will be shown after we learn about loops.

5. Basic arithmetic expressions:

a. Unary operators (one operand): +, -

b. Binary operators (two operands): +, -, *, /, % (addition, subtraction, multiplication, division, remainder)

c. Increment and decrement operators: ++, --

d. Assignment operators: =, +=, - =, *=, /=, %=

```
a = b;        /* assign b's current value to a, implicit cast if necessary */
a += b;       /* shorthand notation for a = a + b; */
```

e. % is for integers only. 10%5 = 0, 7%3 = 1, -7%3 =?, 7%-3=?, -7%-3=?;

f. Precedence (high to low): ( ); + - (unary operators); * / %; + - (binary operators)

4.5+1.47/-2.1 should be evaluated as: 4.5 + (1.47/(-2.1))

-a-b*-c is (-a) - ((-b) * (-c))

g. When the two operands are of the same data type, same will be the result:

1.2+3.1 gives 4.3, 2.0/3.0 gives 0.666667, but 2/3 gives 0

h. Implicit cast: when the two operands have different data type, operation will be done as both are of the type that has higher accuracy:

1 + 3.1 is computed as 1.0 + 3.1, 2/3.0 will be treated as 2.0/3.0

int a = 2;

char b = 19;

short c = 4;

float pi = 3.1416;

example 1: a * b / (c * pi)

example 2: a / b * pi

i. Explicit cast: <*new_type*> *variable_name*

example 3: (double) pi

j. Increment and decrement operators: ++, --

i. x++; is the same as x = x + 1;

ii. x--; is the same as x = x - 1;

iii. x++ vs. ++x

int a = 5, b, c;

```
example 4:  b = a++;        /* a = ?, b = ? * /
example 5:  c = ++a;        /* a = ?, c = ? */
```

iv. ++, -- have the same precedence as unary + and -.

```
example 6:   2 * ++ b + --a      /* this equals ? */
```

6. relational operators: >, >=. <, <=, = =, !=

   a. there is no space between = =, we type a space here on the notes to let you see that there are two = signs. (Recall that = has been used for assignment).

   b. If a relational expression, such as 3 = = 4, is false, the expression has value 0.

   c. If a relational expression, such as 3 != 4, is true, it has some non-zero value (usually 1).

   example 7: in math, we write x+2 < y, $a \geq b$, $3w \leq 5$, $s \neq 4$,

   in C, they are written as x+2 < y, a >= b, 3*w <= 5, s != 4.

   Question: how do we write $a \leq b \leq c$, $|x| \geq 2.718$

7. logical operators: &&, ||, !

   a. &&: logical AND. a && b is true (non-zero) if and only a != 0 and b != 0.

   b. ||: logical OR. a || b is true (non-zero) if a = = 1, or b = =1, or both.

   c. !: logical NOT: !a is true if and only if a = = 0.

   example 8: a <= b && b <= c x >= 2.718 || x <= 2.718

   precedence: ! && || in that order and all are higher than relational operators, which are higher than the assignment (=), which is higher than any of the arithmetic operators.

8. for:

```
for (expr1; expr2; expr3)
{ statements;
}
```

9. if-else

```
if (condition)
{ statements_1;
}
else
{ statements_2;
}
statements_3;
```

10. codes: sizeof.c, cast.c, increment.c

Lab Description

1. Objectives:
    a. Understand the concepts of basic data type and operators.
    b. Learn how to design programs to verify/test (unclear) features in C.
    c. Master basic UNIX commands and its text editors (vi and/or emacs).
2. Pre-lab preparations:
    a. Review lecture notes.
    b. Read and play with the code(s) posted for week 3.
    c. Textbook Reading: sections 3.5 -- 3.10.3, 4.1 -- 4.10.3.
3. In-lab description:

Name: ________________________ Section: 010__ Date: _____________

Lab Report

1. In C, when an integer is divided by another integer, the result will also be an integer. What happens when there is a remainder like in 23/5. Will the result be 4, 5, 4.6, or something else? Write a program to find out the rule (by testing a few examples). Circle the one that you find:

   round to the closest integer   round up   round down

   truncate the integer part   others (specify)

2. In C, the explicit cast operation can change a variable from one data type to another. When we change one from low precision/accuracy to high (for example, from int to float or from float to double), we will not lose any information. What happens if we cast the other way, for example, when casting a float variable to int? Similar to question 1, write a program to find out and write down the rule you find.

3. Insert ()'s for the following expression and give the result:

   5 % 3 * 2 + 12 / 5 - - 3.6 / 9 = __________

4. Work problem 5 (temperature conversion) stated on p. 90 of the textbook.