CMSC 11 Lab Exercises, Week 2

Hi! Welcome back to the ICS labs. Last week we tried editing, compiling and running sample programs without explaining in sufficient detail what the statements in those programs meant.

This week's goals

Understand and apply the concepts of

1) Variables and their types
2) Arithmetic expressions and assignment of values to variables
3) Input and output statements
4) Conditions and if-else branching

Variables and their Types

- variables are used for the temporary storage of values in the computer's memory
- all variables are declared in a program along with their types
- most commonly used types are integers (**int**), floating point numbers with decimal points (**float**), and characters (**char**)
- variable names start with a letter and may be followed by more letters, digits, or the underscore symbol; variable names are case-sensitive in C
- while you can use any name for a variable, the use of a *descriptive name* makes your program self-documenting; sample declarations are shown below

```
int   age;
float inches, cm;
char  firstname[20], surname[20]; /* up to 20 chars long */
```

- basic syntax or format for variable declarations

  type *one or more variables separated by commas*;

Arithmetic expressions and assignment of values to variables

- a value of the appropriate type may be assigned to a variable using an assignment statement, e.g.,

```
int   age;
float inches, cm;
    age = 16;
    inches = 67.8;
    cm = 2.54 * inches;
```

```
float  celsius, fahrenheit;

fahrenheit = 75.0;
celsius = (fahrenheit-32.0) * (5.0/9.0);
```

- basic syntax for an assignment statement: *target variable = expression*;
  This means to evaluate the expression on the right-hand side (substitute the current values of variables present in the expression) and save the result in the target variable named on the left-hand side; if the target variable previously contained some other value, that old value is replaced by the new value. Do not interchange the target variable on the left with the expression on the right.

- arithmetic operators are applied according to precedence rules below:
  multiplications (*), divisions (/) and remainder operations (%) are performed first, left-to-right; additions (+) and subtractions (-) are performed last, left-to-right
  (parentheses are used to change these priorities and improve readability)
- some examples are shown below

expression	*evaluates to the value*
1 + 2 - 3 + 4	4
1 + 2 - (3 + 4)	-4
3.0 + 4.0 / 2.0	5.0
(3.0 + 4.0) / 2.0	3.5
(2 * 3) % 2	0 *(remainder when 6 is divided by 2)*
((1 + 2) * 3) % 2	1 *(remainder when 9 is divided by 2)*
(int) 7/2	3 *(result type casted to int)*
(float) 7/2	3.5 *(result type casted to float)*

- we can not use a normal assignment statement for a character string variable; we need to use a predefined string function named strcpy for the assignment of strings

```
char     firstname[20], middleInitial;

firstname = “Ma. Alice”;      /* wrong string assignment */
strcpy(firstname, “Ma. Alice”);      /* correct version */
middleInitial = 'B';          /* ok for char variables */
```

Input and output statements

- use scanf() to input values from the user
- use printf() to output messages and results of calculations to the user
- format codes: %c for char, %s for string, %d for int, %f for float

– special char code: '\n' represents the newline char for moving to the next line

Example

– some examples of input and output statements are shown in the short but complete program below; note the use of prompts to let the user know that an input is expected
– note that input of variables of simple types (int, float, char) need to be prefixed by an ampersand (&) , but not for character strings

```
$ kwrite convert.c &              use your favorite text editor...
/* a weight conversion program
  by Jim <jimsam@gmail.com>, 22 June 2005
  -- thanks to anduin.eldar.org/~ben/convert/kgram_lb.html  for the conversion formula
*/

#include <stdio.h>

main()
{
 int   age;
 float pounds, kg;
 char  firstname[20];

 printf("your name please? ");       /* ask for name */
 scanf("%s", firstname);

 printf("pls enter your age: ");     /* ask for age */
 scanf("%d", &age);

 printf("your weight in pounds: ");  /* ask for weight */
 scanf("%f", &pounds);

 kg = 0.45 * pounds;                 /* convert lbs to kgs */

 printf("Hi %s.\n", firstname);
 printf("%f pounds is equivalent to %f kg.\n", pounds, kg);
}
```

```
$ gcc -o convert convert.c                    compile...
$ ./convert                    ... and run if there are no errors
your name please? Pedro
pls enter your age: 16
your weight in pounds: 120
Hi Pedro.
120.000000 pounds is equivalent to 54.000000 kg.
```

Improving program readability

- document your program with useful comments /* ignored by the C compiler */
  Some new C compilers (but not all) also allow comments of the form
  // comments of this form are ignored until the end-of-the-line
- use meaningful, self-descriptive variable names
- use indentation, extra spaces and extra lines (but do not abuse)

Conditions and if-else branching

- straight-line programs can be useful but not much interesting; they follow the basic template of a three-step sequence *{ input data; perform calculations; output results; }*
- we can make more interesting programs by performing selected statements only when a condition is satisfied
- syntax: (note: the else clause is optional)

```
if (condition) {
        statements to be performed if the condition is true;
}
else {
        statements to be performed if the condition is false;
}
```

- a condition is a logical (or Boolean) expression which evaluates to either true or false; relational operators are often used for comparing values of expressions

```
== equal          < less than          <= less than or equal
!=  not equal     > greater than  >= greater than or equal
```

- example

```
if (age > 18) { printf("beer or juice? "); scanf(reply); }
else { printf("here's a glass of fruit juice\n"); }
```

Some exercises for you to try on your own

1. Write your own version of a straight-line program that converts a measure in some unit to some other unit. Try converting your age in years to its equivalent in days, hours, minutes and seconds. (Note for Windows/TurboC users: a poor C compiler may not be able to represent very large integers, you may want to use float or long int instead for the type.)
2. Write a temperature conversion program that will allow conversions from celsius to fahrenheit or from fahrenheit to celsius depending on some other user input.
   The following pseudocode should serve as a guide:

   input temperature and scale ('F' or 'C');
   if (scale == 'F') { *convert fahrenheit to celsius and print results* }
   else { *convert celsius to fahrenheit and print results* }

CMSC 11 Lab Exercises, Week 3

Welcome back. Last week we learned the basic foundations of programming, including variables, types and assignment; input and output; and basic arithmetic expressions. We also introduced program branching using the **if-else** statement.

This week's goals

Understand and apply the concepts of branching (also known as selection) and introduce loops (also known as iteration).

The if-else statement

- *Syntax*:

```
if (condition)
   {    statements1;
   }
else
   {    statements2;
   }
```

- *Meaning*: evaluate the condition, if the condition is TRUE, then perform the first group of statements, otherwise perform the second group of statements.
- *Use*: to "program" the computer to perform one or an alternative group of statements depending on some condition.
- the simplest conditions compare two expressions for equality or some similar relation (== equal, != not equal, < less than, <= less than or equal, > greater than, >= greater than or equal)
- note that double-equal sign (==) for testing equality; remember we use a single-equal sign (=) for assignment of values to variables
- Some examples of valid conditions

```
(myAge == yourAge)            (middleInitial != 'X')
(myWeight < yourWeight/2)  (weight/(height*height) >= 30.0)
```

If without an else clause

- the else clause is optional and can be removed if no actions are to be performed if the condition is FALSE; this of course depends on the algorithm logic

```
if (condition)
      {        statements1;
      }
/* else do nothing */
```

Grouping statements together in a branch using { }

- the braces (in either the TRUE branch or the FALSE branch) may be omitted if the group consists of a single statement
- indenting the group(s) of statements help improve readability of program structure; use a tab, or two or more spaces for the indentation

Multiple and nested branching

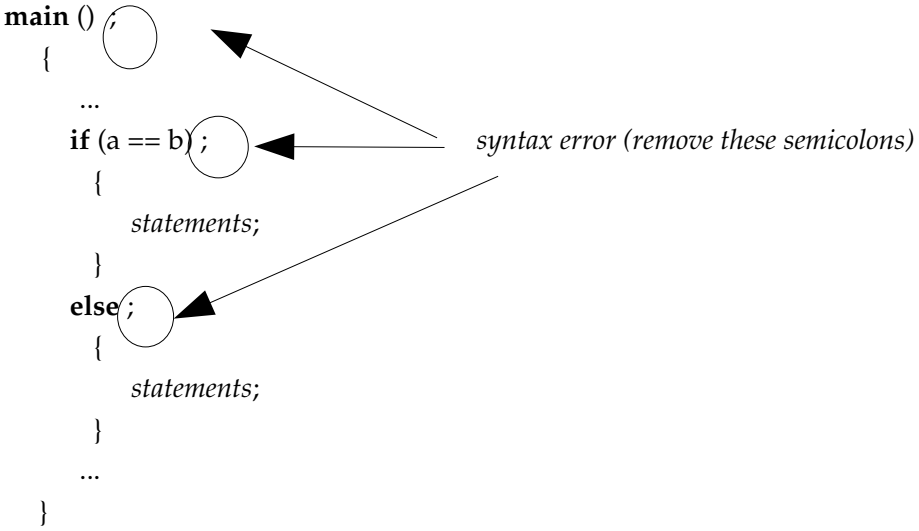
- a branch can have sub-branches that result in multiple branches
-

```
int a, b;

printf("enter 2 integers: ");   scanf("%d %d", &a, &b);
if (a > b)
{
   printf("the first with value %d", a);
   printf(" is bigger than the second with value %d\n", b);
}
else if (a < b)
{
   printf("the first with value %d", a);
   printf(" is smaller than the second with value %d\n", b);
}
else
   printf("A and B both have the value %d\n", a);
```

Watch out for those semicolons

- in C, semicolons are statement terminators, put a semicolon after every assignment, scanf(), and printf() statement; also put a semicolon after every variable declaration
- do not put semicolons after #include <stdio.h> or similar preprocessor directives, after the main() or other similar function headings, nor after the grouping symbols { and }; also do not put semicolons after a (condition) in the if-statement nor after the else keyword



Example.

```
/* Sorting two numbers in ascending order */
#include <stdio.h>
main()
{
   float       a, b;     /* input data */
   float       temp;   /* temporary variable used to swap contents of 2 variables */

   printf("enter 2 numbers: "); scanf("%f %f", &a, &b);
   if (a > b)       /* in wrong order, exchange their values */
   {
      temp = a;            /* temp is necessary to save the original value of variable a */
      a = b;               /* we can now replace the value stored in variable a */
      b = temp;  /* original value of variable of a is now stored in variable b */
   }
   printf("the 2 values in ascending order are %f %f\n", a, b);
}
```

Combining conditions with logical operators

- more complex conditions can be formed using the logical operators AND (denoted by the symbols &&), OR (denoted by ||), and NOT (denoted by !)
- negating a condition with the NOT operator (!) has the effect of reversing the operations to be performed; there are two simple rules to remember

  !(TRUE) evaluates to FALSE

  !(FALSE) evaluates to TRUE

–

- example:

```
if (!(a > b))
        printf ("a is less than or equal to b\n");
else
        printf ("a is greater than b\n");
```

- the *conjunction* of two conditions is TRUE if and only if *both* conditions are TRUE

(TRUE && TRUE) evaluates to	TRUE
(TRUE && FALSE) evaluates to	FALSE
(FALSE &&TRUE) evaluates to	FALSE
(FALSE && FALSE) evaluates to	FALSE

- the dis*junction* of two or more conditions is TRUE if and only if *at least one* of the conditions is TRUE

(TRUE || TRUE) evaluates to	TRUE
(TRUE || FALSE) evaluates to	TRUE
(FALSE ||TRUE) evaluates to	TRUE
(FALSE || FALSE) evaluates to	FALSE

- even more complex conditions can be formed by combining all these into a single condition (conditions are also known as logical expressions or Boolean expressions, after a logician named George Boole)

- precedence rules: NOT (!) first, then AND (&&), and finally OR (||) last; use pairs of parentheses to override these precedence rules as in arithmetic expressions; some examples:

(1==1) && ((1==2) || !(2==3))

is evaluated as	(TRUE AND (FALSE OR (NOT FALSE)))
which is equivalent to	(TRUE AND (FALSE OR TRUE))
or	(TRUE AND TRUE)
or	TRUE

The switch statement for multiple branching

- multiple branching for ordinal data types (**char** or **int**, but not **float**) can also be done using the switch statement; this is an alternative to the cascading **if-else if**-...-**else** combination of statements

```
switch (quizScore)
    {
      case 5:
            printf("excellent!\n");
            break;          /* break is used to get out of the switch statement */
      case 4: case 3:       /* note a branch can have several cases */
            printf("good\n");
            break;
      case 2: case 1: case 0:
            printf("needs improvement\n");
            break;
      default:              /* default clause is optional */
            printf("invalid score\n");
    }
```

Exercises

For numbers 1-5, use any combination of **if** or **if-else** statements to accompish the task.

1. Input any 3 numbers (in random order), and find and print the largest value.
   Hint: One possible algorithm is to do the following:
   First find the larger of the first two items. Then compare the result with the third item.
2. Input any 4 numbers (in random order) and find the average of the two middle values (ignore the minimum and the maximum).
   Hint: Find the smallest and the largest values. Add all four numbers, subtract the minimum and the maximum, and divide by 2.
3. Input any 3 numbers (in random order), and print them in sorted (ascending) order. *Hint*: One possible algorithm is to do the ff.
   { *sort the first adjacent pair; sort the last adjacent pair; sort again the first adjacent pair;* }
4. Enter a temperature in Fahrenheit, convert and print the equivalent temperature in Celsius, and output exactly one of the following messages: "too cold" (< 10C), "too hot" (> 40C), or "just right" (greater that or equal to 10C but less than or equal to 40C).
5. Enter a person's height (in meters) and weight (in kilograms) and compute the body mass index (BMI) according to the formula BMI = weight/height^2. Then output a corresponding message based on the following guide.

BMI	*assessment*
under 18.5	underweight
18.5 - 25.0	normal
25.0 - 30.0	overweight
above 30.0	obese

   Reference: www.consumer.gov/weightloss/bmi.htm