CMSC 104 - Exam 2 Review

General Comments

- Pay close attention to the examples on the slides
- Still need to know earlier topics, e.g. logical and relational operators, though they will not be tested directly.

Loops

- while loop
  - Use for event controlled loops. For example, looping until a user enters a positive number: the program can't know how many attempts the user will need to enter something appropriate.

```
printf("Enter a positive number\n");
scanf("%d", &num);
while (num < 0) {
    printf("Enter a positive number\n");
    scanf("%d", &num);
}
```

  - Another example - reading an unknown number of grades

```
int score, num_scores = 0;
float sum = 0.0, avg;
printf("Enter a student score (-1 to end)\n");
scanf("%d", &score);
while (score >= 0) {
    sum += score;
    num_scores++;
    printf("Enter a student score (-1 to end)\n");
    scanf("%d\n", &score);
}
avg = sum / num_scores;
printf("Average is %f\n", avg);
```

- Correct syntax, especially use of {}.

- do-while loop
  - Use for event controlled loops where you want the body of the loop to execute at least one time.

    In the "enter a positive number" example, we know we want to print the prompt and get input at least one time, so we can replace the while loop with a do-while loop:

```
do {
    printf("Enter a positive number\n");
    scanf("%d", &num);
} while (num < 0);
```

  - Could you replace the while loop in the "entering student grades" example with a do-while loop?
  - Correct syntax, especially use of {}.

- for loop
  - Use for counter-controlled loops, that is, loops that will repeat a known number of times (known to the program a the point where the loop occurs). For example, reading in the test scores for a known, fixed number of students.
  - Parts of a for loop: initialization, test, modification. For example

```
for (i = 0; i < 10; i++) {
    printf("%d\n", i);
}
```

    Prints the numbers 0, 1, ..., 9.

    Initialization is "i=0", test is "i < 10", and modification is "i++".

  - More complicated for loops. How many iterations execute?

```
for (i=0; i<100; i += 5) { ... }
for (i=1; i<=20; i++) { ... }
for (j=1; j<100; j *= 2) { ... }
etc.
```

Assignment Operators

- ++, --
- Difference between pre-increment (++x) and post-increment (x++); also pre-decrement (--x) and post-decrement (x--).
- +=, -=, *=, /=, and %=
- **Be sure to study examples / questions on slides!**

The char Data Type

- Used to store a single character (actually, the integer code representing that character)
- Use of ' ' to denote a character, e.g. `char c = 'X';` assigns the code for the letter X to the variable `c`.
- Use the "`%c`" format string with `scanf()` and `printf()`, e.g. `printf("%c\n", c);`
- The `getchar()` function can be used to read individual characters from the keyboard, but remember that the return key counts as a character.

The Switch Statement

- Use of the **switch** statement and **case**s.
- Proper use of the **default** case
- Proper use of **break**
- Comparison with if and if-else (see slides)
- Example:

```
int day;
...
switch (day) {
     case 0:
          printf("Sunday\n");
          break;
```

```
        case 1:
            printf("Monday\n");
            break;
        case 2:
            printf("Tuesday\n");
            break;
        case 3:
            printf("Wednesday\n");
            break;
        case 4:
            printf("Thursday\n");
            break;
        case 5:
            printf("Friday\n");
            break;
        case 6:
            printf("Saturday\n");
            break;
        default:
            printf("Error - invalid day\n");
    }
```

How would I modify this to print the message "Weekend!" if day is 0 or 6?

Functions

- Three aspects of defining and using a function:
  - Function **prototype** tells the compiler about the function - what arguments it expects and what sort of value it returns.
  - Function **definition** is the code that defines what the function actually does.
  - Function **call** is the point in the program where the function is used.
- A math example:

- "The function *f* has domain the positive real numbers and range the real numbers." This is like the prototype - it tells you that the function *f* expects a positive real input and produces a real output.
- "The function *f* is defined by $f(x) = \ln(x)$." This is like the function definition - it tells us what the function actually does (computes the natural logarithm). Note that *x* is just a placeholder for whatever argument is given to the function.
- "f(3) = 1.0986" *or* "If $y = 7$, then $f(y) = 1.9459$." These are examples of function calls - they *use* the function to compute a value. Note that in math and C, the variable name used in the function definition is just a symbolic placeholder - we can call the function with any value or variable we like.

- Know how to define and use a simple function
  - General syntax, e.g. use of curly braces
  - Declaration of function parameters and their types
  - Declaration of the type of the return value of the function
  - Declaration of the function body
  - Placement of the function prototype
  - Use of the **return** statement within a function body
  - Calling a function correctly
- Parameter passing and local variables
  - Parameter names in the function define the expected inputs to the function. When you call the function with actual constants or variables as arguments, the *values* of those constants or variables are passed to the function. Changes made to the parameters within the function body do not affect the variables in the calling function.
  - For example, the following code prints the value "3":

```
void square(int);
int main() {
     int x = 3;
     square(x);
     printf("%d\n", x);
```

```
        return 0;
}
void square(int x) {
        x = x * x;
}
```

How would you "fix" this code so that it prints the value "9"?

- The purpose and use of header files (e.g. `stdio.h` and `math.h`).