The University of Saskatchewan

Saskatoon, Canada

Department of Computer Science

CMPT 214– Programming Principles and Practice

Assignment 1

Date Due: September 23, 2022, 6pm

Total Marks: 27

General Submission Instructions

- Assignments must be submitted using Canvas.
- Programs must be written in C conforming to the C11 standard. Everything we teach you is compliant with the C11 standard. Things we haven't taught you might not be, so use only the features of C that we have taught. If you try things you find on other websites (where you should definitely **not** be looking for solutions) they may not be C11 compliant. Everything you need to solve programming problems can be found in the course materials, or in the assignment itself.
- Include the following identification in a comment at the top of all your code files: your name, NSID, student ID, instructor's name, course name, and course section number.
- Late assignments are accepted with a penalty depending on how late it is received. Assignments are not accepted more than 2 days late (the assignment submission will close 48 hours after the due date passes and you will not be able to submit). See the course syllabus for the full late assignment and assignment extension policies for this class.
- **VERY IMPORTANT**: Canvas is very fragile when it comes to submitting multiple files. We insist that you package all of the files for all questions for the entire assignment into a single **ZIP** archive file. This can be done with a feature built into the Windows explorer (Windows), or with the **zip terminal command** (LINUX and Mac), or by selecting files in the Mac finder, right-clicking, and choosing "Compress ...". We **cannot accept** any other archive formats other than `ZIP` files. This means no `tar`, no `gzip`, no `7zip` (`.7zip`), and no WinRAR (`.rar`) files. **Non-ZIP archives will not be graded**.
- Instructions on "how to create zip archives" can be found here:
`https://canvas.usask.ca/courses/62306/pages/how-to-zip-slash-compress-your-files`

Question 1 (5 points):

Purpose: To practice console I/O and use of variables.

Write a short story that uses three pieces of numeric data. **The story should be entirely your own. There is no reason why two students should hand in the same story.**

Now pretend that the pieces of numeric data in your story are "blanks" to be filled in by someone else. Write a program that prompts the user to enter the three pieces of numeric data from the console. Print the story to the console with the numeric data read from the console substituted into the appropriate "blanks" in your story.

Sample input and output:

```
Enter an integer: 3
Enter a real number: 2.5
Enter an integer: 9

A young hobbit was walking in the woods about 3 miles from home.
After collecting 2.5 baskets of tasty mushrooms they were beset
upon by 9 restless spirits.  The hobbit took a bite of a mushroom
and offered some to the spirits.  The spirits, their hunger sated,
left the hobbit in peace.
```

Note: the green text in the sample output is the input entered by the user.

Testing

Run your program once, with valid input values. Copy the output and paste it into to a text file called `q1-output.txt` which you will hand in.

Implementation Notes

While you should always strive for good code commenting, commenting will not be assessed for this question.

When reading in numbers using `scanf()`, your program must verify that it actually read a value successfully. Your program must terminate with an appropriate exit code using `errx()` if it does not successfully read a value when one is expected. However, if a value is successfully read, you do not have to check whether the value read is valid (e.g. if the program asked for a positive value, you don't need to verify that it is positive).

If your program runs to completion without encountering any errors, it must return an exit code indicating this.

Question 2 (8 points):

Purpose: To practice use of variables and operators.

In video games, a common trope is that you defeat monsters for experience points. Let's suppose that in some imaginary video game, the player is the hero and they have to defeat monsters to gain experience points to gain levels. In this hypothetical game, the amount of experience that a hero gains for defeating a monster depends on:

- the monster's level; and
- the hero's level.

Let m be the monster's level, and let h be the hero's level. The base amount of experience, xp_{base}, awarded for defeating the monster is:

$$xp_{\text{base}} = 100 + 2.5 \times m$$

The base amount of experience is then adjusted depending on the level difference between the hero and the monster. So the adjusted experience, xp_{adjusted} that is actually awarded to the hero when they defeat the monster is:

$$xp_{\text{adjusted}} = xp_{\text{base}} \times 1.2^{(m-h)}$$

Write a C program that does the following:

1. Print a prompt message to the console asking the user to enter the monster's level, and then read in an integer from the console, storing it in a variable. This is the monster's level.
2. Print a prompt message to the console asking the user to enter the hero's level, and then read in an integer from the console, storing it in a variable. This is the hero's level.
3. Compute xp_{base} for a monster of the level entered by the user.
4. Compute xp_{adjusted} for the hero and monster levels entered by the user.
5. Print to the console a series of messages that report back to the user the following items:
   - The monster's level that was entered in step 1.
   - The hero's level that was entered in step 2.
   - The calculated xp_{base} from step 3.
   - The calculated xp_{adjusted} from step 4.

Testing

Run your program at least twice with different valid values input by the user each time that are different from the sample output, below. Copy the output from all runs and paste it into to a text file called `q2-output.txt`. You'll hand this file in with your code.

Implementation Notes

Your program must terminate with an appropriate error code if a call to `scanf()` fails to read a value. Use the `errx()` library function.

If `scanf()` successfully reads a value, your program may assume that the hero and monster levels entered by the user are between 1 and 50. Your program doesn't have to verify this, and does not have to work correctly for values outside of that range. We can do better than this dangerous assumption, but we are saving that for question 3!

Experience point values should be stored and reported as integers. If at any time you need to convert from a floating-point value to an integer, truncation is acceptable. It is not required to round to the nearest integer, e.g. both 42.8 and 42.2 may be truncated to the integer 42 (this is actually much easier than rounding!).

Remember that to perform the exponentiation operations in the formula, you need the `pow()` function from the math library. In case you forgot, we remind you that Section 5.2.3 of the textbook explains how to use math library functions, and includes a specific example of how to use the `pow()` function. Don't forget to include the `-lm` option when compiling a program that uses the math library (see textbook, bottom of page 59!).

If your program completes successfully, it must terminate with an appropriate exit code.

Sample Output

Here is an example of how your program's output should look. It doesn't need to be identical, but all the same information should be present. **While the output of one run should resemble the sample output, you should perform more rigorous testing than what is shown here.**

```
What is the monster's level?  12
What is the hero's level?  7
The monster is level 12.
The hero is level 7.
The monster's base XP value is 130.
The monster's adjusted XP is 323.
```

Note: the green text in the sample output is the input entered by the user.

Question 3 (14 points):

Purpose: Practice with variables, operators, conditionals, loops, and functions.

This question follows directly from Question 2. Complete question 2 before attempting this question.

For this question we are going to modify our solution to question 2 so that it is more robust to incorrect user input, and supports some additional special monster types. We are going to improve our program so that it can calculate experience point value of not just normal monsters, but also elite monsters and boss monsters.

We will represent monster type as an integer between 1 and 3. The different types of monsters have different formulae for calculating their base experience value. The formula for normal monsters is as before in question 2, while the formulae for elite and boss monsters is different. The following table summarizes the integer value that represents each monster type, and the corresponding base experience point formulae (where m represents the monster's level between 1 and 50 as before).

Monster Type	Integer Representation	xp_{base} Formula
Normal	1	$100 + 2.5 \times m$
Elite	2	$100 + m^{1.2}$
Boss	3	$100 + m^{1.9}$

Starting with your solution to Question 2, modify it to do the following:

1. Write a function that accepts two parameters: an integer representing the monster's level, and an integer representing the monster's type. The function should use the parameters to calculate and return xp_{base} for the given monster type and level. This function may assume that the values passed to the parameters are valid. Us the formulae for xp_{base} in the table above.
2. Write a function that accepts three parameters: the base experience value of a monster, the level of that monster, and the level of the hero. The function should use the parameters to calculate and return xp_{adjusted}, the formula for which is the same as in question 2. This function may assume that the values passed to the parameters are valid.
3. Modify the main program so that when it successfully reads the monster level, if the user entered an invalid value that is not between 1 and 50 (inclusive), it displays an error message, and asks to enter the value again. This process should repeat until the user enters a valid value.
4. Do the same thing for the input of the hero level.
5. Add code that will prompt the user to enter an integer between 1 and 3 indicating the monster's type. If the user does not enter a valid value of 1, 2, or 3 print an error and ask again, repeating this process until the user enters a valid value.
6. Modify the main program so that it uses the functions you wrote in steps 1 and 2 to calculate xp_{base} and xp_{adjusted} instead of computing them directly.
7. Thoroughly comment your code. Use inline comments where appropriate to describe the purpose of small groups of lines that perform specific tasks in both the main program and the two functions from steps 1 and 2. Place a block comment before each of these functions to describe them. Each such comment block should include:
   - a one-sentence description of the function's main purpose;
   - a brief description of the purpose and expected range of values of each parameter; and
   - a description of the function's return value.

Testing

Run your program three times with different valid values input by the user and different from the sample output, below. Each run should input a different monster type, that is, one run should be for a normal monster, one for an elite monster, and one for a boss monster.

Run your program a fourth time, but include invalid integer inputs to demonstrate that your code recovers from such user errors. Copy the output from all runs and paste it into to a text file called `q3-output.txt`. You'll hand this file in with your code.

You do not have to test your program with non-numeric inputs or floating point numbers, but if you wish you may include additional test runs which show your program terminating abnormally under such circumstances.

Implementation Notes

Your program must terminate with an appropriate error code if a call to `scanf()` fails to read a value. Use the `errx()` library function.

Again you'll need the `pow()` function from the math library. Don't forget the `-lm` when compiling!

The program must report the same results as in question 2, that is, the hero and monster levels, and the base and adjusted experience point values for the monster described by the inputs.

Sample Output

Here is an example of how your program's output should look. It doesn't need to be identical, but all the same information should be present. **While the output of one run should resemble the sample output, you should perform more rigorous testing than what is shown here.**

```
What is the monster's level?  90
Error: monster level must be between 1 and 50.
What is the monster's level?  0
Error: monster level must be between 1 and 50.
What is the monster's level?  5
What is the monster's type (1=normal, 2=elite, 3=boss)?  4
Error: monster type must be 1 (normal) 2 (elite) or 3 (boss).
What is the monster's type (1=normal, 2=elite, 3=boss)?  2
What is the hero's level?  100
Error: hero level must be between 1 and 50.
What is the hero's level?  7
The monster is level 5.
The hero is level 7.
The monster's base XP value is: 106.
The monster's adjusted XP is: 73.
```

Note: the green text in the sample output is the input entered by the user.

General Hints

You can expect your programs to go completely wrong when you enter non-numeric inputs. This is not an issue and your programs do not have to recover from such errors (yet). Recovering from non-numeric input to `scanf()` when numeric input is expected is surprisingly difficult. This is because C is very old and low-level, thus, the console I/O functions are not particularly robust or clever. Console I/O with strings in C is notoriously cumbersome, but we'll get there.

What to Hand In

Hand in a `.zip` file archive which contains the following files:

asn1q1.c: The code for your solution for question 1.
q1-output.txt The text file containing the output of your tests of question 1.
asn1q2.c: The code for your solution for question 2.
q2-output.txt The text file containing the output of your tests of question 2.
asn1q3.c: The code for your solution for question 3.
q3-output.txt The text file containing the output of your tests of question 3.

VERY IMPORTANT: Canvas is very fragile when it comes to submitting multiple files. We insist that you package all of the files for all questions for the entire assignment into a single **ZIP** archive file. This can be done with a feature built into the Windows explorer (Windows), or with the **zip terminal command** (LINUX and Mac), or by selecting files in the Mac finder, right-clicking, and choosing "Compress ...". We **cannot accept** any other archive formats other than `ZIP` files. This means no `tar`, no `gzip`, no `7zip` (`.7zip`), and no WinRAR (`.rar`) files. **Non-ZIP archives will not be graded**. We will not grade assignments where these submission instructions are not followed.

Grading Rubric

The grading rubric can be found on Canvas.