The University of Saskatchewan

Saskatoon, Canada

Department of Computer Science

CMPT 214– Programming Principles and Practice

Assignment 2

Date Due: September 30, 2022, 6:00pm

Total Marks: 32

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`

Questions

Question 1 (6 points):

Purpose: To practice for-loops.

For this question you will complete a program that uses the functions for calculating monster experience points for a hypothetical computer game from Assignment 1.

The goal for this problem is to write a program that displays a table of monster experience point values for monsters of a range of levels greater than or equal to an input *starting level* and less than or equal to an input *ending level*. The experience point values will be calculated for a specific fixed input hero level and a specific fixed input monster type (normal, elite, or boss). For a reminder of how monster experience is calculated and how monster type is represented, see Assignment 1, question 3.

Starter code is provided which includes the functions `base_xp()` and `adjuted_xp()` from the solution to assignment 1 question 3. These functions have been improved over their assignment 1 versions. They no longer assume that the values passed to their parameters are legal. Instead they check the validity of their parameters. If they find an invalid parameter value, they use `errx()` to print an error message and terminate the program.

The starter code also includes a `main()` function which prompts the user to input the starting monster level, ending monster level, monster type, and hero level. The given code reads all of these values as integers, but does not verify whether values that are successfully read are valid (though it does terminate if a value cannot be read at all). Fortunately, the new, improved `base_xp()` and `adjuted_xp()` checks for valid values of these inputs for you!

You must complete the program in the `main()` function provided in the starter code so that it does the following:

1. Print a title that indicates the hero level and monster type for the experience point table that is about to be printed (see sample output, below). The monster type should be displayed using its integer representation. The title need not be centered, aligned, or otherwise specially formatted as long as the required information is printed.
2. Print a table with two columns according to the following requirements:
   - The first row of the table should contain column headings. The heading for the first column should be `Monster Level` and the heading for the second column should be `Experience Points (Adjusted)`, as in the sample output below.
   - The first column is the monster level. The row immediately after the header row should be for the monster starting level. The last table row should be for the monster ending level. Rows in between should be for increasing monster levels between these two values — see the sample output. This column must be exactly 15 characters wide with the entries in each row aligned to the **left-hand-side** of the column (see sample output). To make the text align to the left of the column instead of the right, make the minimum width in the format specifier a negative number.
   - The second column contains the adjusted experience point value for the monster of the level in the first column for that row, calculated for the previously input fixed hero level and monster type. Calculate the adjusted experience by calling `adjusted_xp()`. This column must be exactly 30 characters wide with the entries in each row aligned to the **left-hand-side of the column**, as in the sample output.
   - There should be exactly 1 additional space between the two columns. This makes the total width of the table exactly 46 characters.
   - The printing of the non-header rows of your table must be done using a for-loop.

Implementation Notes/Hints

If you're wondering how to force the column widths, have another read of Section 2.5.2 in the textbook. Just remember that to make the column text align to the left instead of the right you need to make the format specifier's minimum width negative (which is not mentioned in the textbook). We also note that negative minimum field widths are used in the `print_ascii_character_codes()` function in code associated with Topic 5, Exercise 5 so it will be instructive to run that program and observe the results.

Testing

Test your program three times, once for each monster type. For each run, use different values for starting and ending monster level and hero level that are also different from the sample output. So that the testing files don't get too large, test cases should output tables containing between 5 and 10 rows of data, not counting the header row. Copy a transcript of the output for all three runs into a text file named `asn2q1-output.txt` and submit it with your program code.

Sample Output

As usual, green text denotes user input.

```
Monster starting level: 5
Monster ending_level: 10
Monster type (1=normal, 2=elite, 3=boss): 2
Hero Level: 1
Generating experience point table for Hero Level = 1, Monster type = 2 ...
Monster Level   Experience Points (Adjusted)
5               219
6               268
7               328
8               401
9               485
10              593
```

Question 2 (4 points):

Purpose: To demonstrate understanding of static and automatic storage allocation.

Consider the provided program, `asn2q2.c`. Duplicate the table below and fill in the blank entries as described below. **There is no coding for this question. Just submit the completed table.**

Variable	Line Declared	Storage Allocation Method	Memory Area	Initial Value	Final Value
`a`	3				
`b`	4				
`c`	5				
`c`	10				
`d`	6				
`w`	12				
`x`	13				
`y`	23				
`z`	24				

The table has one row for each variable declared in the program `asn2q2.c`. The contents of each column are now described:

Variable: The name of the variable.

Line Declared: The line number on which the variable is declared. The line numbers in the table above are the correct line numbers. You can also view a line-numbered version of `asn2q2.c` in the appendix of this document.

Storage Allocation Method: The method of storage allocation that C uses for the variable. Possible entires are **static** or **automatic** (there are no dynamically allocated variables in this program because we haven't learned how to do dynamic allocation yet).

Memory Area The area in the program's memory space that the variable will be stored in. Possible entries are **init.data**, **uninit.data** or **stack** (see Figure 6.1 in the textbook).

Initial Value The initial value that the variable gets **on the line of code where it is declared** or, in the case of global variables, the initial value of the variable when the program begins execution. Possible entries are a numeric value or **undefined**.

Final Value The value of the variable when the program reaches the end of the `main()` function. Possible entires are a numeric value, **undefined**, or **out of scope** indicating that by end of the program the variable has already gone out of scope.

Submit your table as a file named `asn2q2` as a plain text (`.txt`), PDF (`.pdf`) or MS Word (`.doc` or `.docx`) file, or as an image that is a photograph of a handwritten answer in either JPEG (`.jpg`) or PNG (`.png`) format. These are the **only** file formats accepted for this question. Other file formats will be not be opened and receive a grade of zero.

Question 3 (8 points):

Purpose: To assess understanding of basic pointer declarations and operators.

Consider the provided program `asn2q3.c`. Compile it, run it, and observe the output. For each of the variable values that are printed by the main program, that is, `w`, `x`, `y`, and `z`, write few sentences **in your own words** explaining how each of those variables received the final value printed. Your answers should describe the step-by-step changes of values of each variable as execution proceeds through the `main()` and `pointer_stuff()` functions.

For clarity in your explanations, refer to specific line numbers of `asn3q2.c` in your answers. You can view a line-numbered version of `asn2q3.c` in the appendix of this document.

Submit your answers as a file named `asn2q3` as a plain text (`.txt`), PDF (`.pdf`) or MS Word (`.doc` or `.docx`) file, or as an image that is a photograph of a handwritten answer in either JPEG (`.jpg`) or PNG (`.png`) format. These are the **only** file formats accepted for this question. Other file formats will be not be opened and receive a grade of zero.

There is no coding for this question. Just submit your explanation of how each variable obtained its final value.

Question 4 (7 points):

Purpose: To practice using pointers as "output parameters" to functions.

It is difficult to demonstrate the full power and utility of pointers when we have not yet covered things like arrays.

However, one thing we can use pointers for right away is to overcome the limitation that functions can only return one value through their return statement(s). We can use pointers to allow a function to "output" more than one piece of data. Keep in mind that using pointers in this way doesn't actually let the function **return** more than one value, but they can provide an alternative method of getting data out of a function and back to its caller that does not involve a `return` statement.

The concept is quite simple. We can write a function that accepts one or more parameters that are pointers to variables that exist somewhere outside the function. The function can then de-reference those pointers to modify the values of the variables that the pointer parameters "point to". Since the pointer parameters will point to variables that are defined outside the function, the data that the function writes into those variables via the pointers to those variables will be accessible outside of the function.

In this question we will be solving the same problem as in Question 1, but in a slightly different way. Your task is to combine the `base_xp()` and `adjusted_xp()` functions given in Question 1 into a function called `xp()` and use pointers to send both the computed base experience **and** adjusted experience back to the calling function. Your new function is not allowed to actually `return` anything — it must have a return type of `void` and cannot use the `return` statement. The parameters for the new `xp()` function should be:

- the monster level, monster type, and hero level as before;
- two pointer parameters which accept the memory addresses of variables defined outside of the function. Within the function you use an assignment statement that dereferences these pointers to change the values that they point at to the computed base and adjusted xp values. This is how your function will "return" two values without using the actual `return` statement.

The new `xp()` function must retain **all** of the checks for invalid input that are present in `base_xp()` and `adjusted_xp()`. However, it need not duplicate checks that appear in both of the original functions.

The runtime behaviour of the `main()` function in your program should be identical to question 1, but the implementation will change due to the changes to the supporting functions described above.

Implementation Hints

Since you have combined the `base_xp()` and `adjusted_xp()` into a single function, you'll need to call the function to compute the experience points differently compared to question 1. You will also need to reposition and modify the declarations of the variables in `main()` that hold the base and adjusted experience values.

Testing

Use the same tests you used in Question 1 to show that this version of the program produces exactly the same outputs as your solution to Question 1. Copy a transcript of the output for all three runs into a text file named `asn2q4-output.txt` and submit it with your program code.

Sample Output

See the sample output for Question 1.

Question 5 (7 points):

Purpose: To practice switch-case statements.

For this question you'll be simulating a very simple restaurant food menu ordering system.

First, design a unique and exciting menu of at least 6 but no more than 10 delicious food items. Be creative. There's no reason that two students should submit the same menu. Decide on the prices you want to sell each menu item for. Also, decide upon a unique name for your restaurant.

Beginning with the provided starter code, `asn2q5-starter.c`, do the following:

1. Complete the `print_menu()` function so that it prints a numbered list of your menu items and their corresponding prices in addition to what it already prints. Test it to make sure it works by writing a temporary call to `print_menu()` in your `main()` function.
2. Complete the main program. It should do the following:
   (a) Print a welcome message that includes the name of the restaurant.
   (b) Display the menu using `print_menu()` (you need to call the function with an argument that is the current dollar amount of food ordered so far, which is initially zero).
   (c) Get a menu choice using the `get_menu_choice()` function.
   (d) If the menu choice was valid, add the chosen menu item's price to a running total (or end the program if the choice was 0 – see below).
   (e) If the menu choice was not valid, write a message saying that it was not valid.
   (f) In either case, repeat steps (b) through (e) until the user enters 0.
   (g) When the user enters 0, print out a message indicating the total dollar amount of food ordered. The program should then terminate normally. See the sample output below, keeping in mind that your restaurant name and menu will be different.

Implementation Notes

For full marks, the behaviour of `main()` in step 2, above, must be implemented without using any if-statements. This can be accomplished with a combination of a loop and a switch-case-statement.

Names of food items and their prices may be hard-coded literal values. You do not need to store this information in variables.

Testing

Run your program once, inputting at least 3 valid menu choices, and at least one menu choice that is a valid integer but an invalid menu choice. Copy a transcript of the output of this run to a text file named `asn2q5-output.txt` and submit it with your completed program.

Sample Output

As usual, the green text in the sample output is the input entered by the user.

```
Welcome to Aunt Carrie's Seafood Shack
Today's Menu
1. Fried Clams - $15.99
2. Fish & Chips - $9.99
3. Bacon-wrapped Sea Scallops - $18.99
4. Bowl of clam chowder - $5.99
5. Lobster roll - $11.99
6. Extra tartar sauce - $0.79
```

```
So far your order totals $0.00.
Which item should we add to your order (enter 0 if finished)? 1
Today's Menu
1. Fried Clams - $15.99
2. Fish & Chips - $9.99
3. Bacon-wrapped Sea Scallops - $18.99
4. Bowl of clam chowder - $5.99
5. Lobster roll - $11.99
6. Extra tartar sauce - $0.79

So far your order totals $15.99.
Which item should we add to your order (enter 0 if finished)? 4
Today's Menu
1. Fried Clams - $15.99
2. Fish & Chips - $9.99
3. Bacon-wrapped Sea Scallops - $18.99
4. Bowl of clam chowder - $5.99
5. Lobster roll - $11.99
6. Extra tartar sauce - $0.79

So far your order totals $21.98.
Which item should we add to your order (enter 0 if finished)? 5
Today's Menu
1. Fried Clams - $15.99
2. Fish & Chips - $9.99
3. Bacon-wrapped Sea Scallops - $18.99
4. Bowl of clam chowder - $5.99
5. Lobster roll - $11.99
6. Extra tartar sauce - $0.79

So far your order totals $33.97.
Which item should we add to your order (enter 0 if finished)? 9
Sorry, that's not a valid choice!  Try again.

Today's Menu
1. Fried Clams - $15.99
2. Fish & Chips - $9.99
3. Bacon-wrapped Sea Scallops - $18.99
4. Bowl of clam chowder - $5.99
5. Lobster roll - $11.99
6. Extra tartar sauce - $0.79

So far your order totals $33.97.
Which item should we add to your order (enter 0 if finished)? 0
Your order total comes to $33.97.  Have a nice day!
```

Files Provided

asn2q1-starter.c The starter code for question 1. You will also some of this code in question 4.
asn2q2.c The program for which you will be considering variable storage allocation for question 2.
asn2q3.c The program for which you will consider the usage of pointers in question 3.
asn2q5-starter.c The starter code for question 5.

What to Hand In

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

asn2q1.c The code for your solution to question 1.
asn2q1-output.txt The transcript of the tests of your solution to question 1.
asn2q2.XXX Your solution to question 2 where `.XXX` is the appropriate filename extension for the file format you are submitting. For example, if you are submitting a PDF file, submit the file as `asn2q2.pdf`.
asn2q3.XXX Your solution to question 3 where `.XXX` is the appropriate filename extension for the file format you are submitting. For example, if you are submitting a PDF file, submit the file as `asn2q3.pdf`.
asn2q4.c The code for your solution to question 4.
asn2q4-output.txt The transcript of the tests of your solution to question 4.
asn2q5.c The code for your solution to question 5.
asn2q5-output.txt The transcript of the tests of your solution to question 5.

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.

Appendix

```c
#include <stdio.h>

int a;
int b = 100;
static int c;
static int d = 200;

void dostuff() {
    if( c == 0 ) {
        int c = 42;
    }
    static float w;
    static float x = 300;

    printf("w = %f\n", w);
    printf("x = %f\n", x);
    c = 214;

}


int main() {
    float y;
    float z = 400;

    a = 10;
    dostuff();
    d = c / a;

    return 0;
}
```

Line-numbered listing of `asn2q2.c`

```c
#include <stdio.h>

int pointer_stuff(int a, int *b, int *c) {
    *c = a + *b;
    return *c * *c;
}

int main() {
    int x = 2;
    int y = 5;
    int z = 3;
    int *w = &x;

    int result = pointer_stuff(z, &y, &x);

    printf("w points to the value %d, x=%d, y=%d, z=%d, result=%d\n",
           *w, x, y, z, result);
}
```

Line-numbered listing of `asn2q3.c`