Instructions Follow instructions *carefully*, failure to do so may result in points being deducted. Hand in all your source code files through webhandin and make sure your programs compile and run by using the webgrader interface. You can grade yourself and re-handin as many times as you wish up until the due date. Print a hardcopy of the rubric for this assignment and hand it in by the due date.

Partner Policy You may work in pairs for this assignment if you chose. If you do work in any groups or pairs, you must follow these guidelines:

1. You must work on all problems *together*. You may not simply partition the work between you.

2. You should not discuss problem details with other groups or individuals beyond general questions.

3. Hand in only one hard copy (and one soft copy) under the first author’s name/cse login. Be sure to include both names.

Programs

1. In this exercise you will design structures in C that models a savings account and a bank customer that owns the account. You will also write several supporting functions and methods that process and act on that data. Additional details are provided in the `account.h` header file.

The first step will be to design your structures. You will need to decide what fields and data types should be included in this structure. At a minimum, you should support the following pieces of data.

Customer:

- First Name
- Last Name
- Social Security Number

Savings Account:

- Account Number (which may include hyphens and letters)
- Annual Percentage Rate
- Balance
- The customer owning the account

Some items to note:
• The APR will be on the scale \([0, 1]\), so 5\% is 0.05.

• The annual interest should be computed using the Annual Percentage Yield (APY):

\[
e^{\text{APR}} - 1
\]

which is multiplied by the balance.

• All dollar figures are expected to be rounded to whole cents

• When working with generic void pointers in C and using arrays, you cannot simply index using the usual 0, 1, 2, etc. indices. Recall that when elements are stored in an array, the index represents an offset of a memory address. If the array is an array of integers or \texttt{double} or some other build-in type, the compiler knows how large each one is and is able to compute the appropriate offset given the usual 0, 1, 2, etc. indices.

However, when dealing with \texttt{void*} elements, a function must be told how many bytes each element takes, say \texttt{size}. Then each element can be indexed by multiplying an index by the size. That is,

\begin{verbatim}
1 arr[0 * size]  //first element
2 arr[1 * size]  //second element
3 arr[2 * size]  //third element
4 ...
5 arr[i * size]  //i-th element
\end{verbatim}

2. You will redevelop the methods and data structures in Java by finishing Java classes, \texttt{Account.java} and \texttt{Customer.java} that have already been provided for you.