Project 1: Credit Assessment Program

(300 points)

You work as a mainframe application developer for the credit card division of a major bank. Aside from software that processes credit card applications, your group additionally supports day-to-day credit processing activities, including both transactions and updating credit eligibility for existing card holders. You have been tasked with developing a new program that will assess a card-holder's credit-worthiness and update both the variable interest rate and maximum available credit on their account. Your program will use a customer credit information and traction history database as inputs to the algorithms that will decide upon any changes to APR (annual percentage rate) and maximum line of credit.

1 Requirements

Loading Database Tables

Unfortunately, there's no way to communicate with an actual database from within the ASSIST runtime—and no way to set one up and run it on Marist even if it were possible—so we'll have to make do with tables loaded from data members. Your program will therefore need to create its own database tables in program storage, populating them with content provided by a dataset member. The required JCL syntax will be provided for you in a minimal JCL skeleton to get you started. For the sake of simplicity, we'll use just the one dataset member to hold the content of all tables. The end of content for a given table will be marked with a special character ('*') in its own record, for which your program will test to determine when to stop building the corresponding in-storage table.

Your program will first build the customer credit information table, which contains as columns the customer name, billing address, account ID, estimated income, maximum line of credit and credit scores from all three credit bureaus (Equifax, TransUnion and Experian). Your program must use a DSECT to address into the rows of the in-storage table. Keep in mind that your DSECT definition will need to respect fullword boundaries on binary integer data (i.e., the credit scores). Your program will next build the customer transaction history table, which contains as columns the customer's account ID, the category of the transaction type and the amount of the transaction. Transaction types are listed in the algorithms section. The input row specifications are given in section 2.

Algorithms

Your program will then use the customer and transaction database to determine two things: 1) An adjustment to their current APR if the customer's spending habits (comparing payments to expenses) require it; and 2) If the difference between estimated income and payments made to the account, given their credit score, make them eligible for an increase in their line of credit. Note that the algorithms work independently of each other and may be implemented in any order.

APR Adjustment

You must implement an algorithm to calculate adjustments to the customer's APR. This algorithm will use as inputs the payments made to the account, the weighted sum of transactions by type and the customer's maximum line of credit on the account. The transaction types and brief summaries of their impact on the APR is as follows:

- **Living:** Expenses associated with costs of living, such as food, clothing and travel. These will have minimal impact on the APR.
- **Money transfer:** Balance transfers between credit cards or other types of loans. These will have high impact on the APR.
- **Entertainment:** Expenses associated with entertainment, such as movie theater tickets, fees at recreational facilities, gym memberships, and so on. These will have a moderate impact on the APR.
- **Payments:** Payments made to the credit account. These will have moderate impact on the APR.
- **Miscellaneous:** Everything else. These have no impact on the APR.

For each type of transaction, the algorithm should first calculate an average monthly total. It will then compute a weighted sum of the averages over the credit limit, using the formula in Equation 1 below. For coefficients $c_1 \ldots c_4$, refer to the table of coefficients. Note that these are subject to change.

$$\frac{c_1 \cdot \text{avg. living} + c_2 \cdot \text{avg. transfer} + c_3 \cdot \text{avg. entertainment} - c_4 \cdot \text{avg. payment}}{\text{credit line max}} \tag{1}$$

Table 1: Table of APR Coefficients

Coefficient	Value
c_1	.1
c_2	1.8
c_3	.45
c_4	.425

Specifically, your algorithm should compute transaction totals belonging to a given category for each month and then calculate the average. It should do this for each category, and these averages will then become the inputs to the numerator in Eq. 1. Your algorithm should next look at the result—if the computed ratio is greater than .45, then calculate the difference between it and .45 and then multiply by 10 (i.e., $(\text{ratio} - .45) \cdot 10$) to compute the recommended percentage increase. Finally, your program should store the computed APR changes, along with customer account IDs, in a separate table in storage.

A note about fractional coefficients and ratios, and monetary amounts: As we have no way to perform floating point operations, your algorithm will need to perform the calculations using integer division. Recalling that, for example, .45 is equivalent to $\frac{45}{100}$, it should be a fairly simple matter to implement the necessary fractional computations using decimal-to-fraction conversions. Also, *amounts in the transaction data will be given in cents*, not dollars, so your computations will need to take that into account as well.

Line of Credit Increase Recommender

You must also implement an algorithm that will determine if the customer is eligible for an increase in their line of credit. The algorithm will not need to calculate a new limit, but should simply create a list of customers that meet the requirements—we can assume that those customers would be sent on for further review by a credit review committee, but our responsibility ends with flagging them.

The algorithm should begin by first considering the difference between the estimated monthly income and average monthly payment made to the account relative to outgoing expense transactions. Specifically, if the average sum of the monthly outgoing (i.e., non-payment) transactions is within 20% of monetary value of the payments, and if the estimated monthly income is at least 5 times greater than the average monthly payments, then the customer will be considered provisionally eligible for a line-of-credit increase.

The algorithm will then use the customer's credit score data to make a final recommendation. If the customer's average credit score is below 580 then they will not be eligible for a limit raise. If it is above 580, then the algorithm will then determine the degree of the customer's credit-worthiness based on the following scale and store a special code corresponding to the customer's credit rating:

Avg. Score Range	Code	Interpretation
580-669	`CW01`	Marginal credit worthiness
670-739	`CW03`	Good credit worthiness
740-799	`CW04`	High credit worthiness
800-850	`CW07`	Superior credit worthiness

The algorithm should store the recommendations in a separate table in storage, placing into it only the account ID and recommendation code.

Report Generation

After the algorithms are done performing their tasks, your program will then create reports summarizing both (note that *after* does not mean *during*). The first report should, for each customer separately, present summary information regarding the data used to determine any change in their APR. The APR report should list and organize a customer's total transaction amounts by month, followed by the computed APR adjustment. Use the following specification to organize the report:

```
Name: John Doe                                           Account: xxxxxxxxxxxx
Beginning APR: xxxx                                      Credit Limit: xxxxxx
   Ending APR: xxxx

Transaction History Summary:

          Living     Transfer    Entmnt.    Misc.     Payments   Net
==========================================================================
January
February
March
April
May
June
```

```
July
August
September
October
November
December
---------------------------------------------------------------------------
Average
===========================================================================
```

Please note that although a net monthly transaction total is not used in the APR calculations, the report still requires you to compute one; reports are for human consumption, so this is important information for a committee to have if the APR adjustment were to come under review. Monetary amounts given in the report *should not include cents*, so only the dollar amounts are needed here. Your program will therefore have to perform the conversions. You don't need to worry about rounding up to the nearest dollar, so just drop the cents part.

Lastly, the month field in the transaction data is given in the form of an integer value. Your program will need to map integers to the corresponding written equivalent.

The final report that your program will generate will be a recommended credit line increase table. For this part you will simply reproduce the contents of the table your algorithm built in storage as job output. However, as this is intended for a human audience your program will need to retrieve some additional information based on account IDs to fill in the details. Use the following specification to build the output for your report:

```
Credit Increase Recommendations:

Name            Account             Current Limit   Eligibility
===========================================================================
John Doe        xxxxxxxxxx          xxxxxxxx        CWxx
...
===========================================================================
```

Note that both the name and current limit fields should not be kept in the in-storage table built by the algorithm, so you will need to retrieve both from the customer information table (i.e., using the account ID).

2 Input Table Specifications

Refer to the following to determine the number of bytes per column (or field) as table row data is (x)read into the input buffer. Note that column data is always aligned and will occupy exactly the number bytes given below, and that each column is separated from adjacent columns by a single space.

Table 2: Buffered Customer Row Specification

Field	Bytes
Account ID	8
Name	20
Est. Yearly Income	8
Credit Limit	6
Current APR	4
Equifax Score	3
TransUnion Score	3
Experian Score	3

Table 3: Buffered Transaction Row Specification

Field	Bytes
Account ID	8
Month	2
Day	2
Transaction Type	9
Amount	6

3 Coding and Documentation Standards

Assuring that your code follows proper coding conventions and documentation standards is a critical habit in the workplace, and can make the difference between keeping or losing your job. Your work is accordingly graded in part on your ability to adhere to the following coding and documentation standards:

Coding standards

1. **Available registers.** You may only use registers 2 through 11 (inclusive) for loading values, addresses and performing math operations. Registers 0–1 and 12–15 are reserved for special use.
2. **DSECT usage.** Where specified, you must declare and use DSECTs. If the use of a DSECT is suggested, then you are encouraged but not required to use them.

Documentation standards

1. **Documentation box.** Your program code must contain a standard documentation box (or docbox) immediately following the opening JCL statements:

```
*****************************************************************
* CSCI 360-2                PROJECT n                SPRING 2023 *
*                                                               *
* NAME: (Your name)                                             *
* DATE: (Date of submission)                                    *
*                                                               *
* (Summary Description)                                         *
*                                                                *
* REGISTER USAGE:                                                *
*   R2: (Description)                                            *
*   R3: (Description)                                            *
*   ...                                                          *
*                                                                *
******************************************************************
```

n – Number of the current lab exercise (i.e., 1)
a – Lowest register number
b – Next register number

2. **Inline documentation.** Roughly 80% of your lines of code should have comments separated from and following the operands of the Assembler instructions.

If your submitted code violates *any* of these standards, then you will automatically lose 50% of the points ***you would have received***, ***AFTER*** your program's functionality and output are assessed.