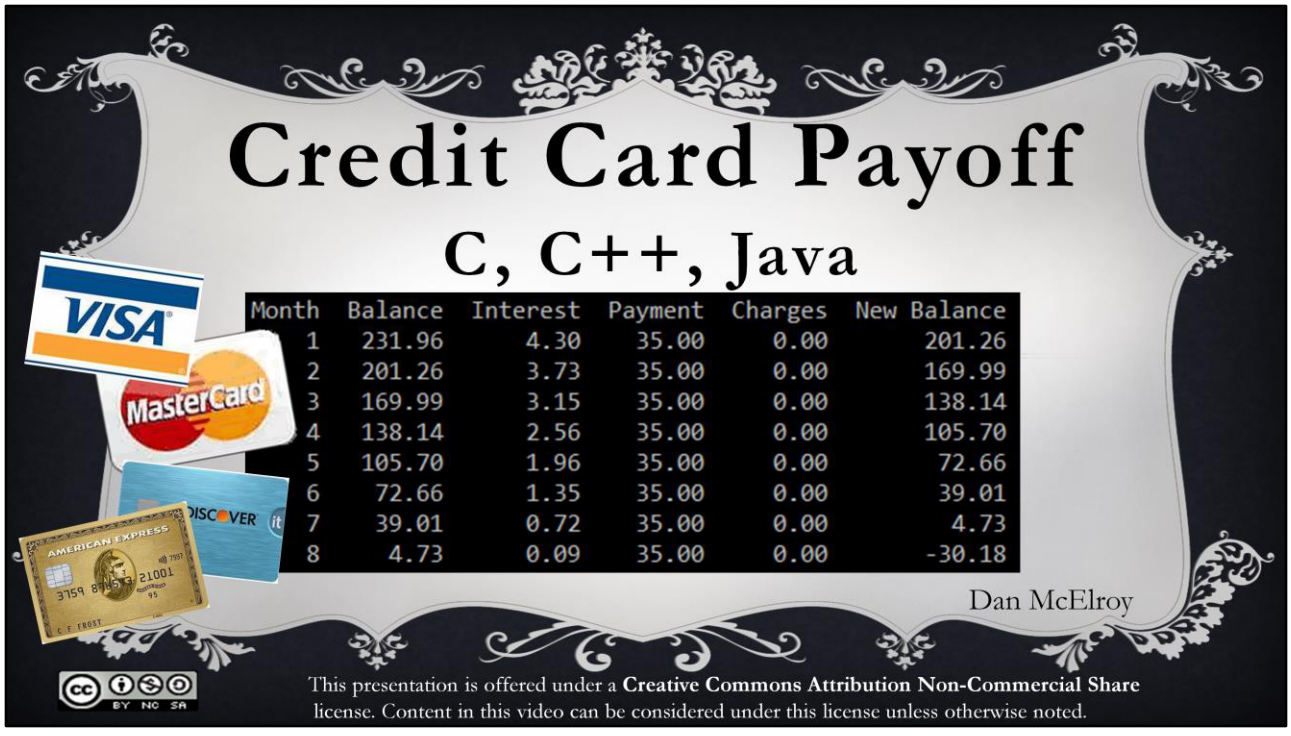
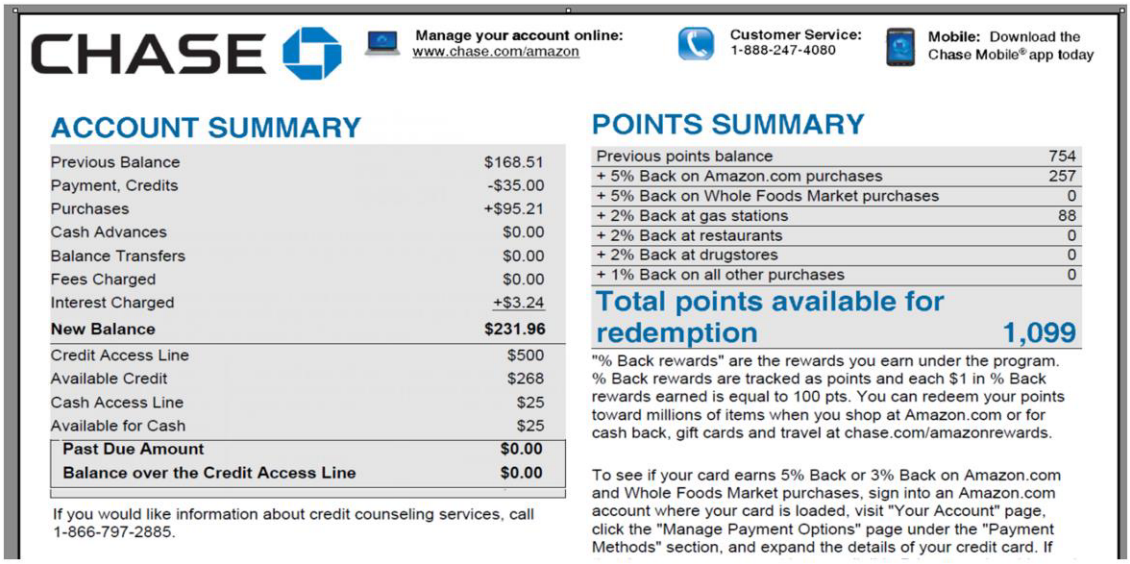


Hello Programmers. I am going use a loop that shows a credit card being paid down and how long it takes depending on the interest rate, any new charges and the amount of each payment.

Code to get started on the project is located at the following locations
C -language version is at
http://program-info.net/C++/downloads/C++CreditCard/C-CreditCard.c
C++ language version is at
http://program-info.net/C++/downloads/C++CreditCard/C++CreditCard.cpp
Java language version is at
http://program-

info.net/Java/downloads/JavaCreditCard/JavaCreditCard.java

Amazon Credit Card Bill



CHASE

Manage your account online: www.chase.com/amazon

Customer Service: 1-888-247-4080

Mobile: Download the Chase Mobile® app today

ACCOUNT SUMMARY

Previous Balance	$168.51
Payment, Credits	-$35.00
Purchases	+$95.21
Cash Advances	$0.00
Balance Transfers	$0.00
Fees Charged	$0.00
Interest Charged	+$3.24
New Balance	**$231.96**
Credit Access Line	$500
Available Credit	$268
Cash Access Line	$25
Available for Cash	$25
Past Due Amount	**$0.00**
Balance over the Credit Access Line	**$0.00**

If you would like information about credit counseling services, call 1-866-797-2885.

POINTS SUMMARY

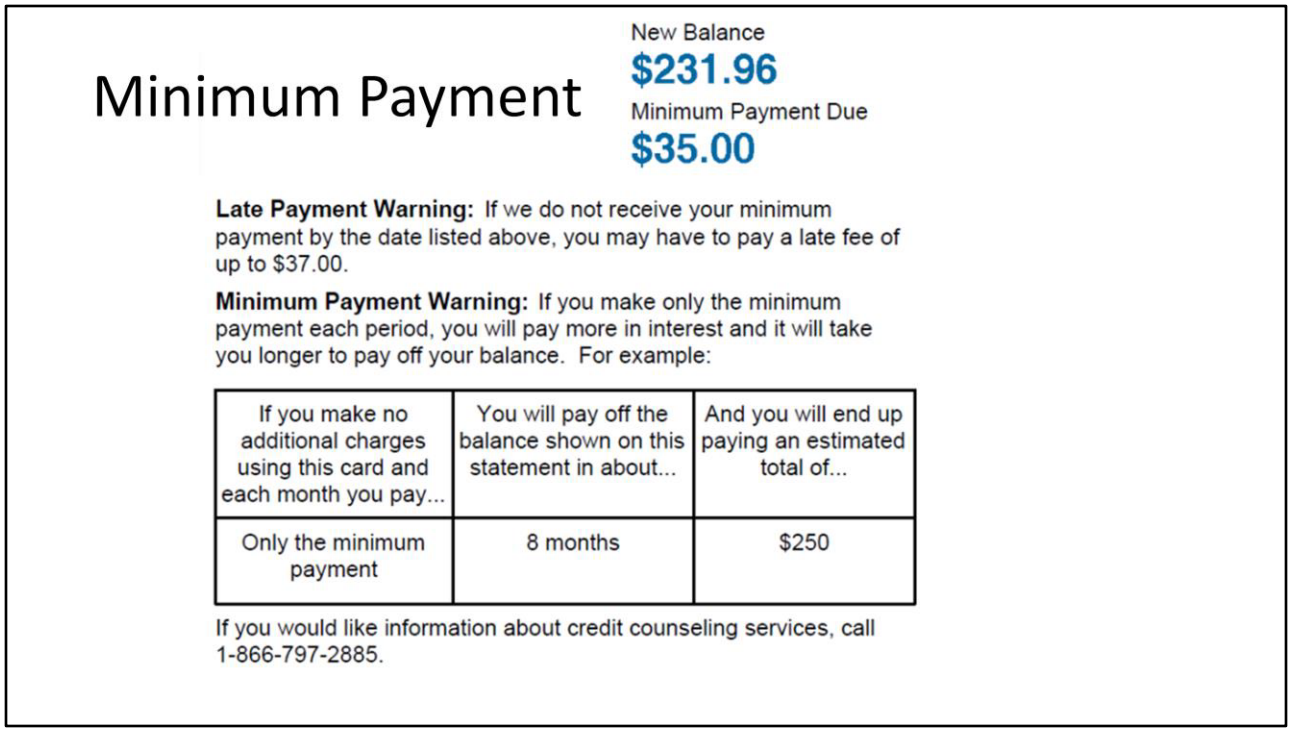
Previous points balance	754
+ 5% Back on Amazon.com purchases	257
+ 5% Back on Whole Foods Market purchases	0
+ 2% Back at gas stations	88
+ 2% Back at restaurants	0
+ 2% Back at drugstores	0
+ 1% Back on all other purchases	0
Total points available for redemption	**1,099**

"% Back rewards" are the rewards you earn under the program. % Back rewards are tracked as points and each $1 in % Back rewards earned is equal to 100 pts. You can redeem your points toward millions of items when you shop at Amazon.com or for cash back, gift cards and travel at chase.com/amazonrewards.

To see if your card earns 5% Back or 3% Back on Amazon.com and Whole Foods Market purchases, sign into an Amazon.com account where your card is loaded, visit "Your Account" page, click the "Manage Payment Options" page under the "Payment Methods" section, and expand the details of your credit card. If

Here is one of the bills from my Amazon credit card issued by Chase Bank. I have deleted some of the personal information, but it still shows my balance of $231.96, minimum payment of $35.00, and current purchases of $95.21 and interest charged of $3.24.

I setup an automatic payment for the minimum amount due so that I won't forget to pay the bill and end up with a late charge. Sometimes I pay extra to help get the bill paid down. I feel really good some months when I get this bill paid in full. But other times I charge more things at Amazon when I still have a balance due.



Minimum Payment

New Balance
$231.96
Minimum Payment Due
$35.00

Late Payment Warning: If we do not receive your minimum payment by the date listed above, you may have to pay a late fee of up to $37.00.

Minimum Payment Warning: If you make only the minimum payment each period, you will pay more in interest and it will take you longer to pay off your balance. For example:

If you make no additional charges using this card and each month you pay...	You will pay off the balance shown on this statement in about...	And you will end up paying an estimated total of...
Only the minimum payment	8 months	$250

If you would like information about credit counseling services, call 1-866-797-2885.

There is a table on the bill that shows "If you make no additional charges using this card and each month you pay only the minimum payment", "You will pay off the balance shown on this statement in about 8 months", "And you will end up paying an estimated total of $250". The current bill is only $231.96 so Chase will charge me about $18 if I space out the payments and don't charge anything more.

Amazon Card Interest Computation

Balance Type	Annual Percentage Rate (APR)	Balance Subject To Interest Rate	Interest Charges
PURCHASES			
Purchases	22.24%(v)(d)	$177.32	$3.24
CASH ADVANCES			
Cash Advances	24.99%(v)(d)	- 0 -	- 0 -
BALANCE TRANSFERS			
Balance Transfer	22.24%(v)(d)	- 0 -	- 0 -

30 Days in Billing Period

(v) = Variable Rate
(d) = Daily Balance Method (including new transactions)
(a) = Average Daily Balance Method (including new transactions)
Please see Information About Your Account section for the Calculation of Balance Subject to Interest Rate, Annual Renewal Notice, How to Avoid Interest on Purchases, and other important information, as applicable.

*Includes interest charges on Late or Return Payment fees.
**This My Chase Loan has expired. Interest will continue to accrue on this My Chase Loan balance until it is paid in full.

The method for computing interest is shown near the bottom of the bill. It shows the annual interest rate of 22.24% and the balance subject to the interest rate is $177.32. Let's see how this is computed.

The interest for a full year would be $177.32 times 22.24%. The 22.24% expressed as a decimal is 0.2224.
So the total would be $39.44 in interest, assuming that the balance remains constant at $177.32.

But this bill is only for a month, or actually 30 days in this billing period. So the interest for this period should be computed at $177.32 * 0.2224 * 30/365 because I am only being charged for 30 days out of the 365 days of a full annual rate.

177.32 * 0.2224 * 30 / 365 = 3.241312 rounded to $3.24 the same as shown on the bill.

To make the program a little easier to write, I am just going to compute interest based on 12 months in a year instead of trying to worry about whether there are 30 or 31 days in a month and then worry about February. In the example programs, I am taking the annual interest rate, dividing it by 12 to get the average monthly interest rate. My computations will show the interest computed a little higher some months and a little lower some months than what appears on my actual statement.

Balance Payoff Table

Month	Balance	Interest Rate	Interest	Payment	Charges	New Balance
1	231.96	22.24%	4.30	35.00	0.00	201.26
2	201.26	22.24%	3.73	35.00	0.00	169.99
3	169.99	22.24%	3.15	35.00	0.00	138.14
4	138.14	22.24%	2.56	35.00	0.00	105.70
5	105.70	22.24%	1.96	35.00	0.00	72.66
6	72.66	22.24%	1.35	35.00	0.00	39.01
7	39.01	22.24%	0.72	35.00	0.00	4.73
8	4.73	22.24%	0.09	35.00	0.00	-30.18

I used a spreadsheet to see what it would look like if I paid the minimum due of $35.00 each month with no additional charges. I am going to compute values each month until the balance reaches $0.00, or in this case it ended up going negative.

Output from the Program

Month	Balance	Interest	Payment	Charges	New Balance
1	231.96	4.30	35.00	0.00	201.26
2	201.26	3.73	35.00	0.00	169.99
3	169.99	3.15	35.00	0.00	138.14
4	138.14	2.56	35.00	0.00	105.70
5	105.70	1.96	35.00	0.00	72.66
6	72.66	1.35	35.00	0.00	39.01
7	39.01	0.72	35.00	0.00	4.73
8	4.73	0.09	35.00	0.00	-30.18

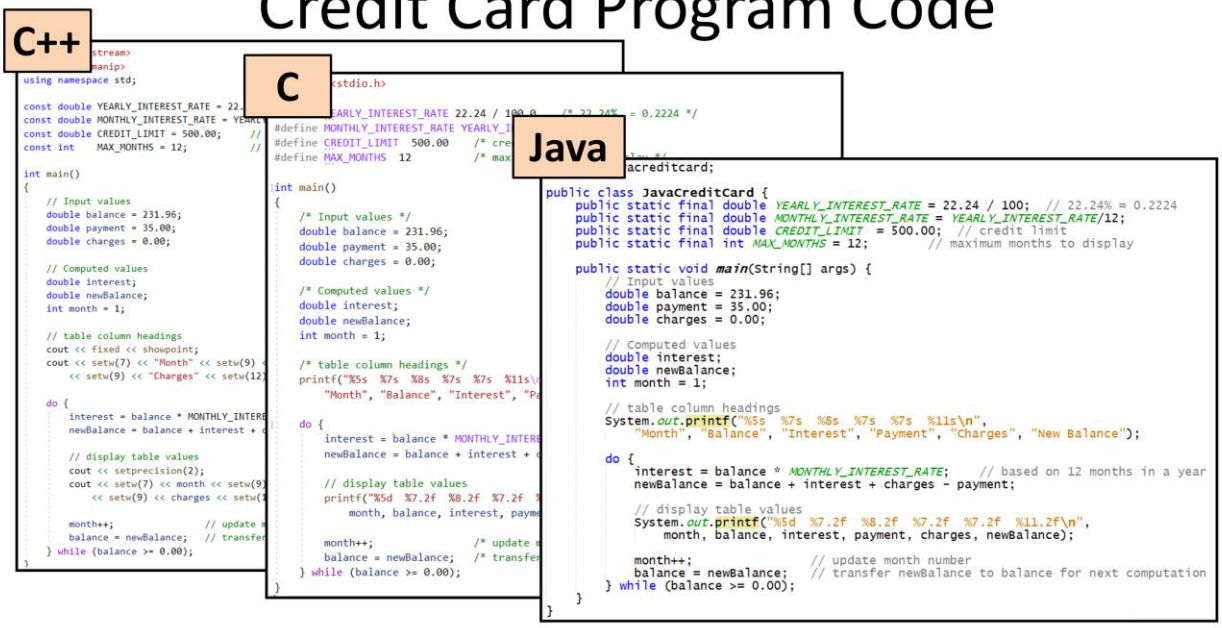
I want to take a close look at the output display produced by the program. It is organized into columns with a header at the top identifying each column. The width of each column is determined by the number of characters for each header. The month column displays a list of months as an integer starting from 1. The rest of the columns display floating-point data with two places past the decimal.

The first entry for Balance is the starting balance originally placed in the program's balance variable.
The interest for each month is computed based on the balance and the interest rate.
The payment each month is $35.00 which is from the payment variable in the program.
At this point, charges are 0.00 each month, even though the

Amazon bill I am showing had $95.21 in charges for that month. I can see that if I am paying only $35.00 each month and I also have additional charges, I may never get this thing paid off.

I see at the end of the 8th month, I should have only paid the balance of $4.73 instead of the $35.00.

Credit Card Program Code



I am going to show the code that creates this table in C++, C and Java. As we review the code, I want you to think about what changes need to be done to implement the following updates.

1) The payment should not cause a negative balance.
2) The loop needs to end if the balance is greater than the credit limit.
3) Do not display more than 12 months, even if there is still a balance on the account.

The only real differences between the C++, C and Java code for this project are the way the constants are defined and the console output statements. C++ uses cout for console output, C and Java use printf. The definition of the variables, the computation of interest, the new balance and the way

the loop is coded are exactly the same for all three versions of the program.

C++ Declaration of the Constants

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

const double YEARLY_INTEREST_RATE = 22.24 / 100.0;    // 22.24% = 0.2224
const double MONTHLY_INTEREST_RATE = YEARLY_INTEREST_RATE / 12;
const double CREDIT_LIMIT = 500.00;     // credit limit
const int    MAX_MONTHS = 12;           // maximum months to display

int main()
{
    // Input values
    double balance = 231.96;
    double payment = 35.00;
    double charges = 0.00;
```

Constants are declared in the C++ version using the const keyword. Constants are declared for YEARLY_INTEREST_RATE, MONTHLY_INTEREST_RATE, CREDIT_LIMIT and MAX_MONTHS. The yearly interest rate is 22.24%. This is the interest rate on the Amazon card that I have. Because it is a percentage, I need to divide by 100 to get the actual decimal value instead of a percentage. The average monthly interest rate is the yearly rate divided by 12.

The CREDIT_LIMIT is defined at 500.00, which is the credit limit for my Amazon card. The credit limit is currently not used in the code that I am providing. You need to use the CREDIT_LIMIT when you update the program to make the loop stop if the balance exceeds the credit limit.

Be careful because if the interest and the new charges were about the same as the payment, the balance would never grow or get paid off and the loop would continue forever the way the program is coded now. MAX_MONTHS is defined as an integer equal to 12. You need to make the loop stop after 12 months even if there is still a balance on the account.

C-language Declaration of the Constants

```c
#include <stdio.h>

#define YEARLY_INTEREST_RATE 22.24 / 100.0    /* 22.24%  = 0.2224 */
#define MONTHLY_INTEREST_RATE YEARLY_INTEREST_RATE/12
#define CREDIT_LIMIT  500.00    /* credit limit */
#define MAX_MONTHS  12          /* maximum months to display */

int main()
{
    /* Input values */
    double balance = 231.96;
    double payment = 35.00;
    double charges = 0.00;
```

The #define preprocessor directive is used to declare the constants in the C-language. We need to remember that with the #define, it really becomes a text find and replace in the code for the remaining part of the program. There is no data type associated with the #define, there is no = assignment operator, and we need to make sure that we don't put a semicolon ; as part of the #define or it will be also become part of text replacement when it is probably not desired. I even made that mistake when converting the program from C++ to C.

Java Declaration of the Constants

```
package javacreditcard;

public class JavaCreditCard {
    public static final double YEARLY_INTEREST_RATE = 22.24 / 100;  // 22.24% = 0.2224
    public static final double MONTHLY_INTEREST_RATE = YEARLY_INTEREST_RATE/12;
    public static final double CREDIT_LIMIT  = 500.00;  // credit limit
    public static final int MAX_MONTHS = 12;        // maximum months to display

    public static void main(String[] args) {
        // Input values
        double balance = 231.96;
        double payment = 35.00;
        double charges = 0.00;
```

The constants in Java are declared using the public static final directives with a data type, the name of the constant and a value assigned to the constant.

C++ Display the Data

```
Month   Balance   Interest   Payment   Charges   New Balance
    1    231.96       4.30     35.00      0.00        201.26
```

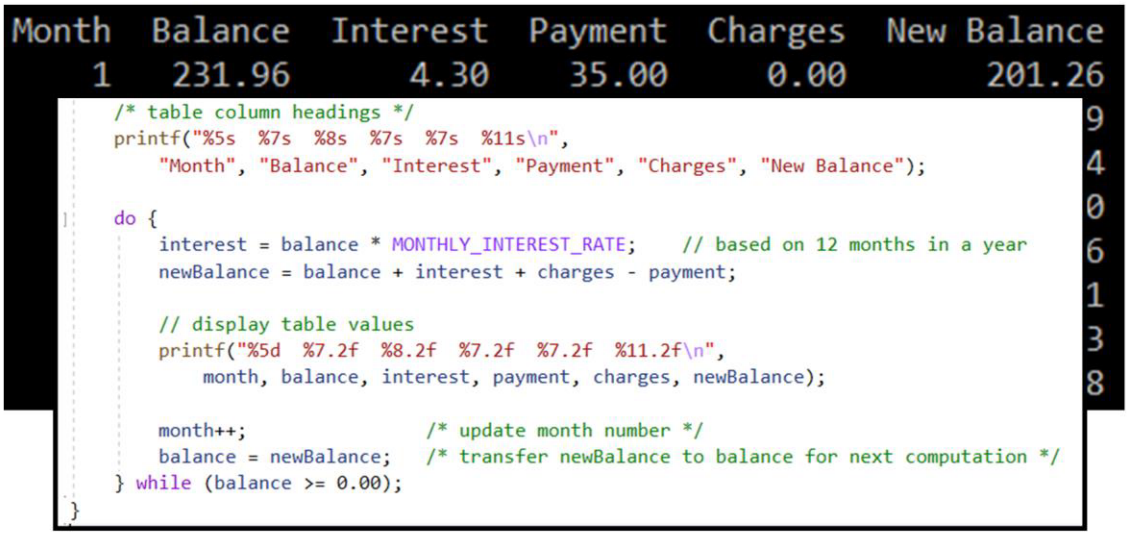
```
    // table column headings
    cout << fixed << showpoint;
    cout << setw(7) << "Month" << setw(9) << "Balance" << setw(10) << "Interest" << setw(9) << "Payment"
        << setw(9) << "Charges" << setw(12) << "New Balance" << endl;

    do {
        interest = balance * MONTHLY_INTEREST_RATE;
        newBalance = balance + interest + charges - payment;

        // display table values
        cout << setprecision(2);
        cout << setw(7) << month << setw(9) << balance << setw(10) << interest << setw(9) << payment
            << setw(9) << charges << setw(12) << newBalance << endl;

        month++;                // update month number
        balance = newBalance;   // transfer newBalance to balance for next computation
    } while (balance >= 0.00);
}
```

The widths of the columns on the display are set to the number of characters for each of the column headings plus two extra character positions to space the columns apart. For example, the first column heading title "Month" is 5 characters. C++ is using the setw(7) for the first column. By using the same setw( ) value for each column in the heading as well as during the table display inside the do loop, the columns will line up very nicely. The first column is 7 characters wide, the next columns are 9, 10, 9, 9, and 12 characters wide.

The cout statements for the table header are literal character strings but the cout statements inside the do while loop that display the table values are from the numeric values from the variables. For example, the header

for the first column is the literal character string "Month" enclosed in double quotes. But the numeric data in the first column inside the do while loop is from the integer variable named month, which increments from 1 to 8 as the program loops. The cout << setprecision(2); causes two digits past the decimal to be displayed for floating point variables, here declared as double data types. Since the month variable is declared as an integer, it won't have any digits past the decimal.

C and Java Display the Data Using **printf**



Since both the C-langage and Java use printf to display data to the console, it was really easy to provide code for both of these languages. Once I converted the C++ program to C using printf instead of cout statements, all I needed to do when updating printf statements in the Java program is add System.out in front of each printf statement.

The C-language uses printf and Java uses System.out.printf. Very cool.

When using printf, it is just easier to set each column the desired number of spaces and then add the two extra spaces as part of the format string.

The table header is displaying strings. Therefore, the format

string in printf has "%5s %7s %8s   etc.
The format string is followed by the strings that fill in the %5s %7s %8s etc. Those are "Month", "Balance", "Interest", "Payment", etc.

The printf statement inside the do…while loop is displaying integer data for the month and floating point data with the double data type for the rest of the columns. The format string starts with "%5d %7.2f %8.2f etc. The %5d says to display a decimal integer with a width of 5 character positions. The %7.2f says to display floating point data using 7 character positions that include 2 digits past the decimal. Because I used field widths of 5, 7, 8, 7, 7 and 11 for both the heading row and the data rows, everything should line up with the same size columns for the header and the data rows.

Interest Computation

```
    // table column headings
    cout << fixed << showpoint;
    cout << setw(7) << "Month" << setw(9) << "Balance" << setw(10) << "Interest" << setw(9) << "Payment"
        << setw(9) << "Charges" << setw(12) << "New Balance" << endl;

    do {
        interest = balance * MONTHLY_INTEREST_RATE;
        newBalance = balance + interest + charges - payment;

        // display table values
        cout << setprecision(2);
        cout << setw(7) << month << setw(9) << balance << setw(10) << interest << setw(9) << payment
            << setw(9) << charges << setw(12) << newBalance << endl;

        month++;                // update month number
        balance = newBalance;   // transfer newBalance to balance for next computation
    } while (balance >= 0.00);
}
```

```
Month   Balance   Interest   Payment   Charges   New Balance
    1    231.96       4.30     35.00      0.00        201.26
    2    201.26       3.73     35.00      0.00        169.99
```

The newBalance is computed as the balance + the monthly interest + charges - payment. I used a new variable named newBalance so that I would be able to display both the current balance and newBalance before the balance is updated at the start of the next month.

Here is a hint for all three versions of the program. The way the program is currently coded, it shows a negative balance at the end of the 8th month. One of the requirements of the project is that the payment should not be more than the balance. I recommend creating a new variable and naming it balanceBeforePayment.

Change the line newBalance = balance + interest + charges - payment; into two lines

```
balanceBeforePayment = balance + interest + charges;

newBalance = balanceBeforePayment - payment;
```

In between those lines, use if statement to decide to just pay the balance if the balance is less than the standard payment or pay the regular payment. You may need to think this one through to get it right.

Test for the Credit Limit

```c
/* Input values */
double balance = 231.96;
double payment = 35.00;
double charges = 100.00;

/* Computed values */
double interest;
double newBalance;
int month = 1;

/* table column headings */
printf("%5s  %7s  %8s  %7s  %7s  %11s\n",
    "Month", "Balance", "Interest", "Payment", "Charges", "New Balance");

do {
    interest = balance * MONTHLY_INTEREST_RATE;    // based on 12 months in a year
    newBalance = balance + interest + charges - payment;

    // display table values
    printf("%5d  %7.2f  %8.2f  %7.2f  %7.2f  %11.2f\n",
        month, balance, interest, payment, charges, newBalance);

    month++;                /* update month number */
    balance = newBalance;   /* transfer newBalance to balance for next computation */
} while (balance >= 0.00);
```

Month	Balance	Interest	Payment	Charges	New Balance
1	231.96	4.30	35.00	100.00	301.26
2	301.26	5.58	35.00	100.00	371.84
3	371.84	6.89	35.00	100.00	443.73
4	443.73	8.22	35.00	100.00	516.96

If I charged an additional $100.00 but only paid $35.00 each month, the balance on the card would never get paid off. It wouldn't take long to exceed the credit limit of $500.00.

Part of the project definition is 2) The loop needs to end if the balance is greater than the credit limit.

Currently, the do...while loop keeps looping as long as the balance >= 0.00. Now I want to add an additional condition to keep looping as long as the balance <= CREDIT_LIMIT.

Although it might look good for an expression in a math class to say

  while (0.00 <= balance <= CREDIT_LIMIT);

this won't work in programming. We need to code it as

follows

```
  while (balance >= 0.00 && balance <= CREDIT_LIMIT);
```

The && is the AND operator and needs a boolean true/false value on each side of the &&.
balance >= 0.00 is a comparison operator that produces a boolean true/false result.
balance <= CREDIT_LIMIT is also a comparison operator that produces a boolean true/false result.

As far as order of precedence, the comparison operators > >= < <= == and != have higher precedence than the && AND logical operator. Therefore, we don't need a bunch of parentheses ( ) around the different parts of the expression.

Limit the Display to 12 Months

```c
/* Input values */
double balance = 231.96;
double payment = 35.00;
double charges = 30.00;

/* Computed values */
double interest;
double newBalance;
int month = 1;

/* table column headings */
printf("%5s  %7s  %8s  %7s  %7s  %11s\n",
    "Month", "Balance", "Interest", "Payment", "Charges", "Ne

do {
    interest = balance * MONTHLY_INTEREST_RATE;    // based on 12 months in a year
    newBalance = balance + interest + charges - payment;

    // display table values
    printf("%5d  %7.2f  %8.2f  %7.2f  %7.2f  %11.2f\n",
        month, balance, interest, payment, charges, newBalance);

    month++;                /* update month number */
    balance = newBalance;   /* transfer newBalance to balance for next computation */
} while (balance >= 0.00);
```

Month	Balance	Interest	Payment	Charges	New Balance
1	231.96	4.30	35.00	30.00	231.26
2	231.26	4.29	35.00	30.00	230.54
3	230.54	4.27	35.00	30.00	229.82
4	229.82	4.26	35.00	30.00	229.08
5	229.08	4.25	35.00	30.00	228.32
6	228.32	4.23	35.00	30.00	227.55
7	227.55	4.22	35.00	30.00	226.77
8	226.77	4.20	35.00	30.00	225.97
9	225.97	4.19	35.00	30.00	225.16
10	225.16	4.17	35.00	30.00	224.34
11	224.34	4.16	35.00	30.00	223.49
12	223.49	4.14	35.00	30.00	222.64

If I charged an additional $30.00 each month and made a payment of $35.00, by the time interest was added each month, it would take over 8 years before the account was paid off. Part of the project definition states 3) Do not display more than 12 months, even if there is still a balance on the account.

This is an easy fix. Update the expression in the while statement with another && AND to loop only while the month <= 12.

The End