**Project 7**

* **Setup**

First we set up the basics for any Unix programming language -

**#!/usr/bin/perl**

**use strict;**

**use warnings;**

Next we set up our home “menu” interface -

**Sub main {**

**while (1) { … } }**

This infinite while loop will make it so that once the user completes one task, they will return to the main menu. Now to put things on the main menu -

**print “Menu:\n”;**

**print “1 - Add Patient\n”;**

**print “2 - End Shift\n”;**

**print “3 - Exit\n”;**

**print “Please enter selection (1-3) - ”;**

**my $choice = <STDIN>;**

**chomp($choice);**

This advises the user to input a number from 1 to 3, which will bring them to that specific task based on the number they input. We can validate this input with an if statement -

**if ($choice == 1) { new\_patient(); }**

**elseif ($choice == 2) { end\_shift(); }**

**elseif ($choice == 3) { last; }**

**else { print “Invalid” }**

If we input 1, the function “new\_patient” would begin. If we input something that was not valid, it would display “Invalid” and have us do it again. Now lets head to add a new patient

* **Input Patient Data**

**sub new\_patient {**

The goal of this function is to input, calculate, and collect all the necessary data for each patient

The first section of this function is mainly for the user to input raw data, for example -

**print “Last Name - “;**

**my $last\_name = <STDIN>;**

**chomp($last\_name);**

The program is prompting the user for their info, which they will put into a variable, in this case “last\_name”. Most data needed can be input with little retooling, but there are two values which do more retooling - the patients date and PIN number.

**my $age = calculate\_age($dob);**

**my $pin = generate\_pin();**

* **Calculate Age and PIN**

Lets start with age. We ask the user for their date of birth in the form of “YYYY-MM-DD”, the separation by dash symbols will help us to separate this single string input into a month, day, and year value to help calculate the patients current age in years.

**my ($dob) =@\_;**

**my ($year, $month, $day) = split('-', $dob);**

**my ($current\_year, $current\_month, $current\_day) = (localtime)[5,4,3];**

The assigning of dob assigns the dob data we brought in as argument from the previous function. The assignment of year month and day splits this data into 3, and assigns the 3 pieces to the three variables. The localtime line assigns the current year(5), month(4) and day(3) values.

**$current\_year += 1900; | $current\_month += 1;**

**my $age = $current\_year - $year**

Both += lines have to do with how localtime works. Localtime only returns the amount of years its been since 1900, meaning 2023 would be displayed as 123. Adding 1900 to this number solves the problem. Localtime also displays the current month more mechanically as well, going off of a range from 0 to 11 rather than 1 to 12. Adding 1 solves this issue. To calculate the users total age, we subtract the patients birth year from the current year to get our patients age.

**$age-- if ($current\_month < $month || ($current\_month == $month && $current\_day < $day));**

**return $age;**

Once our age data has been checked over, we can return it to our new\_patient function. Now we can generate a pin number for the patient, a random number within 9000, plus 1000 -

**sub generate\_pin { return int(rand(9000)) + 1000; }**

* **Display Patient info**

Once all patient details have been input, we can put them into an array. An array to hold all of our patients.

**my @patients;**

We can then make an entry into this array -

**my %patient = (last\_name => $last\_name, first\_name => $first\_name,**

**middle\_initial => $middle\_initial, dob => $dob,**

**insurance => $insurance, ailment => $ailment,**

**age => $age, pin => $pin,);**

To validate the entry into the array, we can use the push command to push out %patient entry into out @patients array

**push @patients, \%patient;**

We can then print out each @patients entry thus far -

**foreach my $patient\_ref (@patients) {**

**my %patient\_info = %{$patient\_ref};**

**my $full\_name = "$patient\_info{first\_name} $patient\_info{middle\_initial} $patient\_info{last\_name}";**

**# (and so on)**

For each entry patent\_info entry in patient\_ref (referring to the patient's array), each individual patient's info will be displayed.

* **Process Patient Data**

At the end of the day, the user can enter into the end\_shift function to process all patient entries from that day. Any patients without insurance will be printed out. Following the printing of patient info similar to what we had just done, we must now check for any patients with no insurance.

**if ($patient->{insurance} eq INSURANCE\_NONE) {**

**print "\*\*\* Billing Department: Patient without insurance \*\*\*\n"; }**

Translation - If this user entry’s insurance info is put in as “none”, the following message will display.

Most notable thing here is the “eq INSURANCE\_NONE” section. “eq” can be used in perl to compare two strings. It is being compared to INSURANCE\_NONE, which is a variable made constant at the beginning of the program -

**use constant { INSURANCE\_NONE => 'none', };**

After all is said and done, we activate the main function by calling it at the tail end of the code -

**main();**