COMP2401 - Assignment #3

(Due: **Wednesday, February 28th @ 11pm**)

In this assignment, you will gain experience in using **structs** and **pointers**. Make sure to put comments in your code … especially ones to describe your functions, definitions and important parts of your code. You must also write your code as cleanly & efficiently as possible.

Assume that a private cell phone company wanted you to write a program to keep track of various calls made between its customers. You will use various structures to represent the type of phone plan, the customer, the calls made and the phone network itself. A header file called **phoneNetwork.h** has been provided with the necessary **struct** definitions that you will need as shown below … with type name definitions shown in red:

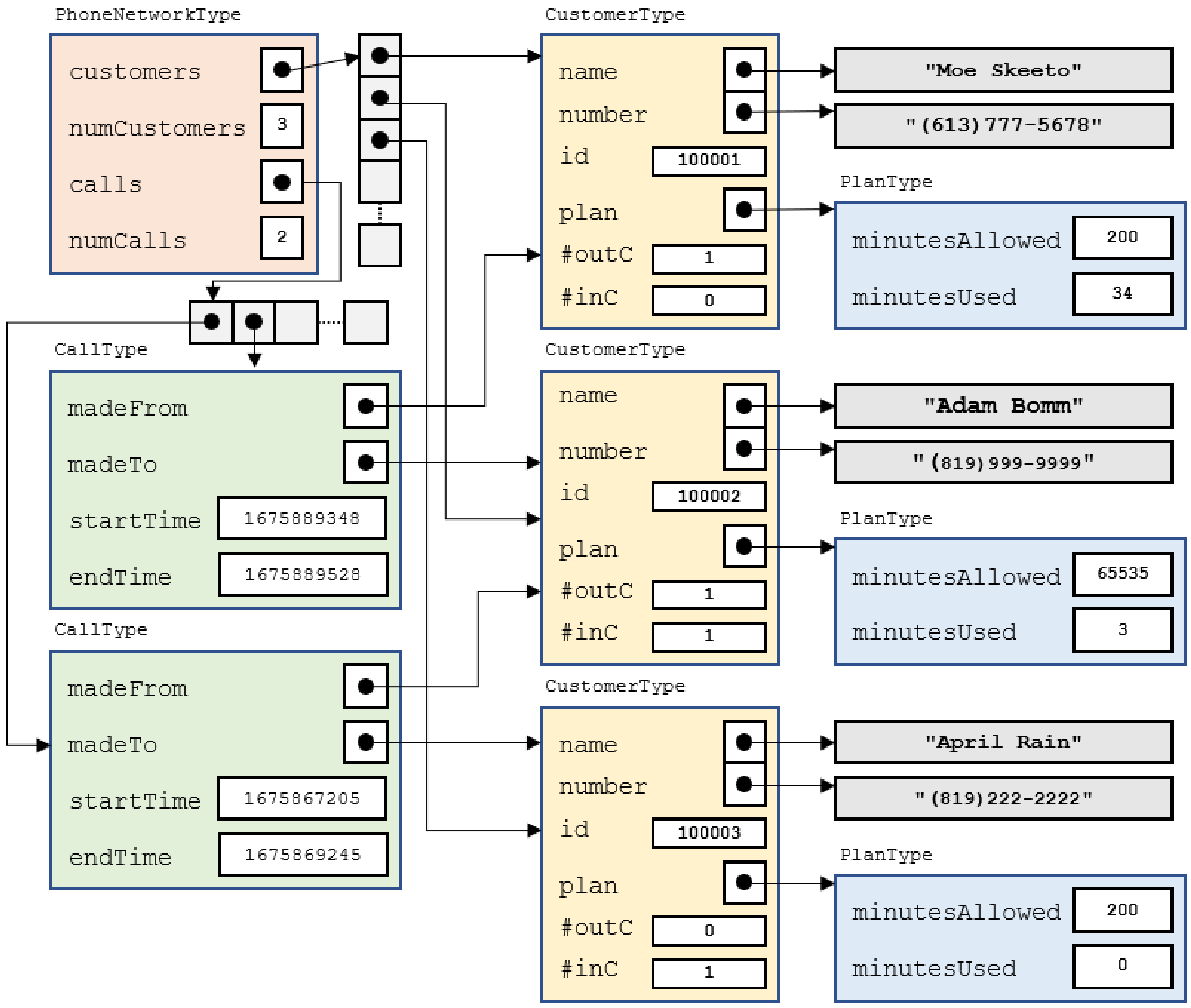
```
// Structure that stores a phone plan that a customer has
typedef struct {
  unsigned short int   minutesAllowed;   // # mins/month allowed in phone plan
  unsigned short int   minutesUsed;      // # mins used this month so far
} PlanType;


// Structure that stores a customer
typedef struct {
  char              *name;         // name of customer
  unsigned int       id;           // customer ID
  char              *number;       // phone number of customer
  PlanType           plan;         // phone plan of customer
  unsigned short     numOutCalls;  // number of calls made this month
  unsigned short     numInCalls;   // number of calls received this month
} CustomerType;


// Structure that stores a phone call that was made
typedef struct {
  CustomerType   madeFrom;   // customer who made call
  CustomerType   madeTo;     // customer who was called
  time_t         startTime;  // time that call was started
  time_t         endTime;    // time that call ended
} CallType;

// Structure that stores the phone network's customer and call data
typedef struct {
  CustomerType customers[MAX_CUSTOMERS];  // data for customers registered with network
  unsigned int numCustomers;              // number of customers registered
  CallType     calls[MAX_CALLS];          // data for all calls that took place this month
  unsigned int numCalls;                  // number of calls that took place this month
} PhoneNetworkType;
```

The next page shows an example of a phone network with **3** customers in which **2** phone calls were made. Make sure that you understand how the image matches up with the structures in the header file before you continue the assignment.



Download the **phoneSimulation.c** file. It contains a **main()** function with comments that indicate the steps you should follow. It also contains sample names, numbers and phone plan values that you will use. You will be writing a bunch of functions and procedures.

You must create a file called **phoneNetwork.c** that will contain ALL the functions and procedures that you will be asked to write. DO NOT write them in the **phoneSimulation.c** file.

You will need to include that additional file at the end of your **gcc** shell command so that it gets compiled along with the **phoneSimulation.c** file.

(1) Write a **registerCustomer()** procedure that will register a customer in the phone network defined by the **phoneNetwork** variable that is provided. The procedure must have the following signature which takes a pointer to a phone network, a name, a phone number and a number of allowed phone plan minutes/month:

```
registerCustomer(PhoneNetworkType *, char *, char *, int)
```

For each customer you should have some kind of global counter so that each time you register a customer they get assigned a unique id (starting with 100001). It must return **1** if the customer was able to be added (i.e., not past capacity) and **0** otherwise.
Call this procedure from the **main()** function so that it creates exactly **20** customers … with the names, phone numbers and the allowed number of phone plan minutes provided in the Sample arrays.

(2) Write a **displayCharges()** procedure that takes a pointer to a **PhoneNetworkType** and then displays all customers in the phone network provided … showing their id, phone number, name and phone plan minutes allowed. It should show "UN" for unlimited-minute phone plans and "PPM" for phone plans that require paying per minute of usage. Call this procedure at the end of the **main()** function as shown by the comment in the code. You will add more functionality to this procedure later. Make sure to test your code … the output should be aligned nicely and approximately as follows:

```
 ID     Phone Number     Name               Plan
-------------------------------------------------
 100001 (613)111-1111    Rob Banks            UN
 100002 (819)222-2222    April Rain          200
 100003 (613)333-3333    Rita Book           PPM
 100004 (343)444-4444    Sue Permann          UN
 100005 (819)555-5555    Tim Bur             PPM
 100006 (613)666-6666    Paddy O'Lantern      UN
 100007 (343)777-7777    Sam Pull            200
 100008 (613)888-8888    Sandy Beach          UN
 100009 (819)999-9999    Adam Bomm            UN
 100010 (613)555-1234    Hugo First           UN
 100011 (613)555-5678    Dan Druff           200
 100012 (613)666-1234    Mabel Syrup          UN
 100013 (613)666-5678    Mike Rohsopht        UN
 100014 (343)777-1234    Adam Sapple         100
 100015 (613)777-5678    Moe Skeeto          200
 100016 (819)888-1234    Anita Bath           UN
 100017 (343)888-5678    Rusty Chain         PPM
 100018 (613)999-1234    Stu Pitt            200
 100019 (613)999-5678    Val Crow             UN
 100020 (613)444-1234    Neil Down            UN
```

(3) Write a **resetMonth()** procedure that simulates the resetting of phone call data for a new month of calls. It must take a pointer to a **PhoneNetworkType**. The procedure should reset everything as if there have been no calls made in the network so that things begin from scratch. Call this procedure in your **main()** function right after creating the customers.

(4) Write a **recordCall()** function that simulates a call that has been made in the network. It must have the following signature, taking a pointer to the network as its 1st parameter:

```
char recordCall(PhoneNetworkType *, char *, char *, time_t, time_t);
```

The 2nd parameter is the phone number that the call was made from, the 3rd parameter is the phone number that the call was made to, the 4th parameter is the time that the call was started and the last parameter is the time that the call ended. The time is designed as a **time_t** type. This is a large integer number that represents the number of seconds that has elapsed since 00:00:00 UTC, January 1, 1970. It is the typical standard that is used to record time-of-day in computer science.

The function should return **0** if the call could not be recorded (this could happen if the number of calls that were made in the phone network has reached its capacity or if either of the phone numbers are invalid). Otherwise, the function should return 1. To record the call, you will need to do a few things. You will need to:

(a) Determine how many minutes the call lasted (rounded up to the nearest minute)

(b) Find the customer that made the call and record that number of minutes used as well as the fact that this customer made this outgoing phone call.

(c) Find the customer that received the call and record the fact that this customer received this incoming phone call. If it was a PAY_PER_MINUTE customer that received the call, the number of minutes used should be reduced as a result of this phone call, but other types of phone plans do not need to have the minutes reduced for incoming calls.

(d) Update the appropriate from/to/start/end information in the network's **calls** array accordingly.

(5) To test the **recordCall()** function, after resetting the month in the **main()** function, you should write code to make **10** calls (we will increase this to **100** later). To do this, choose a random customer to make the call and a random one to receive the call (make sure that it is not the same customer that makes and receives the call at the same time as this is invalid). You will need to choose a random start time for the call and a random end time. To do this, we will need to generate realistic **time_t** values. There is a `struct tm` structure defined in `<time.h>` that allows us to specify an exact year, month, day, hour, minute and second time. Here is what the structure looks like:

```
struct tm {
  int tm_sec;          // seconds 0-59
  int tm_min;          // minutes 0-59
  int tm_hour;         // hours 0-23
  int tm_mday;         // day of the month 1-31
  int tm_mon;          // month 0-11
  int tm_year;         // year since 1900
  int tm_wday;         // day of the week 0-6
  int tm_yday;         // day in the year 0-365
  int tm_isdst;        // daylight saving time (we will use -1 to disable)
};
```

You should generate a start time for the phone call by making a variable of type **struct tm** and then setting its values so that the call was made in **Feb 2024**. The day of the month should be chosen randomly as a number from **1-29** (since 2024 is a leap year). The hour, minute and second of the day should all be chosen randomly.

To get your **struct tm** variable into a **time_t** format that you need for the **recordCall()** function, you simply call the **mktime()** function from `<time.h>` with the address of your struct tm variable. That function will return the value you need for your call's start time. As for the call's end time, you can take that start time value and simply add a random number of seconds so that the call lasts at most two hours (i.e., at most **120** mins *** 60** seconds). At this point, you will not know if your code works properly until you write the next procedure.

(6) Write a **displayCallLog()** procedure that displays the phone call data. It must take a pointer to a **PhoneNetworkType**. For each call made, the procedure should display the phone number that the call was made from, the number it was made to, the start date/time of the call, the end date/time of the call and the length of time (in minutes with one decimal place) that the call lasted. The output should be formatted neatly and look roughly as follows:

```
Call From #     Call To #       Start Time:          End Time             Call Length
-------------------------------------------------------------------------------------
(613)555-1234   (613)333-3333   2024-02-07 12:57:50   2024-02-07 13:19:32    21.7 minutes
(613)999-1234   (613)333-3333   2024-02-25 05:34:07   2024-02-25 07:26:46   112.7 minutes
(613)444-1234   (819)999-9999   2024-02-15 08:20:22   2024-02-15 09:59:54    99.5 minutes
(613)555-1234   (343)444-4444   2024-02-16 09:12:20   2024-02-16 09:52:32    40.2 minutes
(613)555-1234   (819)222-2222   2024-02-16 03:13:00   2024-02-16 03:24:41    11.7 minutes
(343)888-5678   (613)999-5678   2024-02-12 18:55:09   2024-02-12 20:35:15   100.1 minutes
(343)777-7777   (613)111-1111   2024-02-21 02:23:53   2024-02-21 03:40:39    76.8 minutes
(343)777-7777   (613)999-5678   2024-02-28 13:33:57   2024-02-28 14:51:59    78.0 minutes
(343)777-1234   (613)555-1234   2024-02-12 18:23:27   2024-02-12 18:54:04    30.6 minutes
(343)888-5678   (613)444-1234   2024-02-06 12:49:37   2024-02-06 12:50:00     0.4 minutes
```

Note: To extract the date/time information from the **time_t** values, you will need to get the **time_t** value into a struct tm variable so that you can extract what you need. To convert a **time_t** value called **timeValue** into a `struct tm` variable called **daytime**, you can call the **localtime()** function as follows: `struct tm daytime = *localtime(&timeValue);`

Don't forget to adjust the year and month values accordingly after the conversion.

(7) Finally, we will adjust the **displayCharges()** procedure so that it computes and displays the appropriate monthly charges for each customer. Here is how the charges work:

(a) For customers with UNLIMTED minute phone plans, the base cost is **$34.99** per month (before taxes).

(b) For customers with phone plans that have a specified number of minutes, the base cost is **$9.99** per month. But there is an extra **15** cents charge for every minute that the customer went over the plan's allowed minutes. So, a customer that used up **126** minutes on a 100-minute phone plan would pay **$9.99** + (**$0.15** * **26**) = **$13.89** before taxes.

(c) For customers with a PAY_PER_MINUTE phone plan, the customer simply pays **$0.15** per minute for the first **100** minutes and then they pay **$0.35** for every minute over the **100**.

Adjust your **displayCharges()** procedure so that it also displays (alongside the current customer information) the number of outgoing calls made, the number of incoming calls received, the number of minutes used, the number of minutes gone over, the base cost, the extra cos (for going over), the HST amount and the final total. The information must be aligned and displayed neatly. Here is an example of the output:

```
Phone Number     Name                Plan Out  In     Used     Over       Base      Extra        HST       Total
---------------------------------------------------------------------------------------------------------------
(613)111-1111    Rob Banks             UN   3   5      234        0      34.99       0.00       4.55    $  39.54
(819)222-2222    April Rain           200   6   3      303      103       9.99      15.45       3.31    $  28.75
(613)333-3333    Rita Book            PPM   2   7      563      463      15.00     162.05      23.02    $ 200.07
(343)444-4444    Sue Permann           UN   6   9      382        0      34.99       0.00       4.55    $  39.54
(819)555-5555    Tim Bur              PPM   4   4      367      267      15.00      93.45      14.10    $ 122.55
(613)666-6666    Paddy O'Lantern       UN   8   5      729        0      34.99       0.00       4.55    $  39.54
(343)777-7777    Sam Pull             200   4   6      246       46       9.99       6.90       2.20    $  19.09
(613)888-8888    Sandy Beach           UN   6   7      329        0      34.99       0.00       4.55    $  39.54
(819)999-9999    Adam Bomm             UN   4   3      280        0      34.99       0.00       4.55    $  39.54
(613)555-1234    Hugo First            UN   4   3      295        0      34.99       0.00       4.55    $  39.54
(613)555-5678    Dan Druff            200   4   6      170        0       9.99       0.00       1.30    $  11.29
(613)666-1234    Mabel Syrup           UN   3  10      165        0      34.99       0.00       4.55    $  39.54
(613)666-5678    Mike Rohsopht         UN   5   2      343        0      34.99       0.00       4.55    $  39.54
(343)777-1234    Adam Sapple          100   5   2      327      227       9.99      34.05       5.73    $  49.77
(613)777-5678    Moe Skeeto           200   5   6      341      141       9.99      21.15       4.05    $  35.19
(819)888-1234    Anita Bath            UN  10   4      752        0      34.99       0.00       4.55    $  39.54
(343)888-5678    Rusty Chain          PPM   5   7      757      657      15.00     229.95      31.84    $ 276.79
(613)999-1234    Stu Pitt             200   4   5      285       85       9.99      12.75       2.96    $  25.70
(613)999-5678    Val Crow              UN   5   2      209        0      34.99       0.00       4.55    $  39.54
(613)444-1234    Neil Down             UN   7   4      488        0      34.99       0.00       4.55    $  39.54
```

You MUST ensure that you are not using "magic numbers" in your code. You should add appropriate definitions to **phoneNetwork.h** for all cost amounts minute-limit values and tax amounts.

Make sure that your code compiles, runs and produces the correct output. Hand this test code in with your assignment.

IMPORTANT SUBMISSION INSTRUCTIONS:

Submit all your **.c** and **.h** source code files as individual files (i.e., DO NOT **TAR/RAR/ZIP/COMPRESS** anything):

1. A **Readme** text file containing
   - your name and studentNumber
   - a list of source files submitted

    - any specific instructions for compiling and/or running your code
2. All of your **.c source** files and all other files needed for testing/running your programs.

The code **MUST** compile and run on the course VM. You WILL lose marks on this assignment if any of your files are missing. So, make sure that you hand in the correct files and version of your assignment. You will also lose marks if your code is not **written neatly with proper indentation and containing a reasonable number of comments.**