Street Simulator

Jon Winter

Bianca Ebanks

Romanov Andre

5/10/2017

Project Description

This a step by step guide that allows you to create a real life street simulator in the C programming language. This program utilizes real data that was collected from multiple streets (Pine, Dorrance, Washington, Eddy, Mathewson) downcity in Providence, Rhode Island. While creating this street simulator, you will learn how to efficiently use queues through the help of nodes, pointers, and headers. Specifically, in this project you will learn how to enqueue/dequeue cars (nodes) into streets (headers). Overall, which will create a street simulator that represents real data of traffic intensity.

Specify the Problem

A street simulator that allows cars to enter/leave onto multiple streets as a way to represent an accurate traffic intensity.

Criteria	**Constraints**
Cars will enter/leave streets form a first-in first-out fashion which makes the use of a queue logically perfect.	Cars in the real world don’t leave a street straight away. To make our simulator more lifelike we should create a timer that allows cars to leave (dequeue) from one street to another.
The enqueue/dequeue functions should utilize a parameter of the street (header). Which removes the reason to have multiple enqueues/dequeues.	We have multiple streets (headers) however, at this moment an enqueue/dequeue function only accounts for one street. We need to find a way to make the these functions more flexible.
Add an integer called timer to the streets (header) to figure out how long it takes a car to leave (dequeue) a street.	Street lights will play a big part to make this simulator more lifelike. In the real world street lights operate on a timer. The simulator could also use this concept and make the street lights an integer.
Create three extra functions that relies on a chance factor and accounts for streets that may simply dequeue or dequeue onto one or more streets.	

Gather Data

The data that was collected by the groups will be used to give us a rough idea of the traffic intensity.

A street (header) will have a maximum amount of cars that can fit else it will create a traffic jam and no more cars will be able to fit on that street.

A street (header) will also have a timer (integer variable) which will be the amount of time it takes for a car to leave (dequeue) a street.

Once the simulator starts there will already be cars on the streets, however, with a function that relies on chance the simulator will allow more cars to enter (enqueue) the streets.

Analysis

The program needs to be able to:

Enqueue/Dequeue for multiple streets (header).

Account for a timer as way to know when each car is leaving.

Account for another timer as a way to know the street lights.

Account for the maximum amount of cars that can fit on a street and check for a traffic jam.

Print a statistic of an accurate representation of the traffic intensity.

Implementation

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 20

typedef struct node
{
   int data;
   struct node *next;
} node;

typedef struct head
{
   struct node *front;
   struct node *rear;
   int count, timer, lightCount, TTL, COC;
} head;

void metaupdate(struct head *, int, int);
void enqueue(struct head*);
void startup();
int dequeue(struct head *header);
void chance2(struct head *, struct head *);
void chance3(struct head *, struct head *, struct head *);
void movecar(struct head *, struct head *, struct head *, struct head *);
void display (int, int, struct head *,struct head *,struct head *,struct head *,struct head *,struct head
*,struct head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct
head *,struct head *,struct head *,struct head *,struct head *);
void regularTraffic(struct head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct
head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct head
*,struct head *,struct head *,struct head *,struct head *);
void specialEvent(struct head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct
head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct head
*,struct head *,struct head *,struct head *,struct head *);
void menu(struct head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct head
*,struct head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct head *,struct
head *,struct head *,struct head *,struct head *);

int main()
{
```

```
struct head pineA;
metaupdate(&pineA, 4, 2);

struct head pineB;
metaupdate(&pineB, 2, 0);

struct head richmond;
metaupdate(&richmond, 2, 1);

struct head weyA;
metaupdate(&weyA, 1, 2);

struct head weyB;
metaupdate(&weyB, 2, 2);

struct head weyC;
metaupdate(&weyC, 3, 2);

struct head westA;
metaupdate(&westA, 3, 2);

struct head westB;
metaupdate(&westB, 3, 1);

struct head washA;
metaupdate(&washA, 3, 0);

struct head washB;
metaupdate(&washB, 3, 1);

struct head dorA;
metaupdate(&dorA, 2, 3);

struct head dorB;
metaupdate(&dorB, 2, 3);

struct head dorC;
metaupdate(&dorC, 2, 0);

struct head empireA;
metaupdate(&empireA, 2, 1);

struct head empireB;
```

```
   metaupdate(&empireB, 2, 1);

   struct head empireC;
   metaupdate(&empireC, 2, 0);

   struct head eddy;
   metaupdate(&eddy, 2, 1);

   struct head mat;
   metaupdate(&mat, 2, 1);
   startup();
   menu(&pineA, &pineB, &richmond, &weyA, &weyB, &weyC, &eddy, &mat, &westA, &westB,
&washA, &washB, &dorA, &dorB, &dorC, &empireA, &empireB, &empireC);
   return 0;
}
void startup()
{
   printf ("\t\tPROVIDENCE TRAFFIC MODELING SYSTEM \n");
   printf("\t\t\t\t(PTMS)\n\n\n\n");
   printf("\t\t\tDevelopers:\n\n");
   printf("\t\t\t\tBianca Ebanks\n\t\t\t\tJonathan Winter\n\t\t\t\tRomanov Andre\n\n\n\n\n\n");
   system("pause");
}
void menu(struct head *pineA,struct head *pineB,struct head *richmond, struct head *weyA, struct head
*weyB, struct head *weyC, struct head *eddy, struct head *mat, struct head *westA,struct head *westB,
struct head *washA, struct head *washB, struct head *dorA, struct head *dorB, struct head *dorC, struct
head *empireA, struct head *empireB, struct head *empireC)
{
   int select;

   system("cls");
   printf("Select from the following options: \n\n");
   printf("\t\t1 - Regular Traffic\n");
   printf("\t\t2 - Special Event\n");
   printf("\t\t3 - Exit\n");
   scanf("%i",&select);

   if (select==1)
   {
      regularTraffic(pineA, pineB, richmond, weyA, weyB, weyC, eddy, mat, westA, westB, washA,
washB, dorA, dorB, dorC, empireA, empireB, empireC);
      menu(pineA, pineB, richmond, weyA, weyB, weyC, eddy, mat, westA, westB, washA, washB,
dorA, dorB, dorC, empireA, empireB, empireC);
```

```
    }
    if (select==2)
    {
        specialEvent(pineA, pineB, richmond, weyA, weyB, weyC, eddy, mat, westA, westB, washA,
washB, dorA, dorB, dorC, empireA, empireB, empireC);
        menu(pineA, pineB, richmond, weyA, weyB, weyC, eddy, mat, westA, westB, washA, washB,
dorA, dorB, dorC, empireA, empireB, empireC);
    }
    if (select==3)
    {
        exit(0);
    }

}

void metaupdate(struct head *header, int leave, int chance)
{
    header->front = NULL;
    header->count = 0;
    header->timer = 0;
    header->TTL = leave;
    header->COC = chance;
}

void enqueue(struct head *header)
{
    struct node *newptr;
    struct node *temp;
    newptr = (struct node *)malloc(sizeof(struct node));


    if (header->front == NULL)
    {
        newptr->data = 5;
        newptr->next = NULL;
        header->front = newptr;
        header->rear = newptr;
        header->count++;

    }
    else
    {
        newptr->data = 6;
        newptr->next = NULL;
        temp = header->rear;
        header->rear = newptr;
        temp->next = newptr;
        header->count++;
    }
}

int dequeue(struct head *header)
{
    struct node *temp;

// Empty Queue
    if (header->front == NULL)
    {
        printf("Cannot Dequeue from an empty list\n");
        return(1);
    }
// One Structure in the Queue
    else if (header->front == header->rear)
    {
//printf("Dequeueing from a queue with only one item...\n\n");
        temp = header->front;
        header->front = NULL;
        header->rear = NULL;
// Save the address of tempPtr
        header->count--;
        return(temp);
    }
    else
    {
//printf("Dequeueing from a queue that has more than one item...\n\n");
        temp = header->front;
// = temp->forward
        header->front = header->front->next; //might replace below code
//header.front = head.front->forward;
        header->count--;
    }
}


void chance2(struct head *e, struct head *e2)
{
    int r;

    r = rand() % 2;
    if (r == 0)
    {
        enqueue(e);
    }
    else
    {
        enqueue(e2);
    }
}

void chance3(struct head *e, struct head *e2, struct head *e3)
{
    int r;

    r = rand() % 3;
    if (r == 0)
    {
        enqueue(e);
    }
    else if (r == 1)
    {
        enqueue(e2);
    }
    else
    {
        enqueue(e3);
    }
}

void movecar(struct head *header, struct head *e, struct head *e2, struct head *e3)
{
    if (header->count>0)
    {
        if (header->timer == header->TTL)
        {
            dequeue(header);

            if (header->COC == 1)
            {
//only other way to go
```

```
            enqueue(e);
         }
         else if (header->COC == 2)
         {
            if (e2 == NULL)
            {
            }
            else
            {
               chance2(e, e2);
//enqueue(e);
//enqueue(e2);
            }
         }
         else if (header->COC == 3)
         {
            if (e2 == NULL || e3 == NULL)
            {
            }
            else
            {
               chance3(e, e2, e3);
            }
         }

         header->timer = 0;
      }
      else
      {
         header->timer++;
      }
   }
}

void specialEvent(struct head *pineA,struct head *pineB,struct head *richmond, struct head *weyA,
struct head *weyB, struct head *weyC, struct head *eddy, struct head *mat, struct head *westA,struct
head *westB, struct head *washA, struct head *washB, struct head *dorA, struct head *dorB, struct head
*dorC, struct head *empireA, struct head *empireB, struct head *empireC)
{
   int totalCars=0, x, pineAp, pineBp, weyAp, weyCp, richmondp, weyBp, westAp, westBp, matp, eddyp,
empireAp, empireBp, empireCp, dorAp, dorBp, dorCp, washAp, washBp, whilecnt, totalBefore=0,
carleft=0;
```

```
// Add cars to Pine A
    for (x = 0; x < MAX; x++)
    {
        enqueue(pineA);
    }

// Add cars to Dor A
    for (x = 0; x < MAX; x++)
    {
        enqueue(dorA);
    }

// Add cars to Empire A
    for (x = 0; x < MAX; x++)
    {
        enqueue(empireA);
    }

// Add cars to Wash B
    for (x = 0; x < MAX; x++)
    {
        enqueue(washB);
    }

// VARIABLES
    totalCars = pineA->count + pineB->count + weyA->count + weyC->count + richmond->count +
weyB->count + westA->count + westB->count + mat->count + eddy->count + empireA->count +
empireB->count + empireC->count + dorA->count + dorB->count + dorC->count + washA->count +
washB->count;
    printf("Simulation Start Total Cars: %i\n\n", totalCars);



// SIMULATION ENGINE
    while (totalCars != 0)
    {
// USED LATER TO PRINT BEFORE AND AFTER
        system("cls");
        pineAp = pineA->count;
        pineBp = pineB->count;
        weyAp = weyA->count;
        weyCp = weyC->count;
        richmondp = richmond->count;
        weyBp = weyB->count;
        westAp = westA->count;
        westBp = westB->count;
        matp = mat->count;
        eddyp = eddy->count;
        empireAp = empireA->count;
        empireBp = empireB->count;
        empireCp = empireC->count;
        dorAp = dorA->count;
        dorBp = dorB->count;
        dorCp = dorC->count;
        washAp = washA->count;
        washBp = washB->count;
// LIGHT HERE
        movecar(pineA, pineB, richmond, NULL);
        movecar(pineB, NULL, NULL, NULL);

        movecar(richmond, weyC, NULL, NULL);
        movecar(weyA, weyB, dorA, NULL);
// LIGHT HERE
        movecar(weyB, weyA, NULL, NULL);
        movecar(weyC, empireC, NULL, NULL);
// LIGHT HERE
        movecar(westA, westB, eddy, NULL);
        movecar(westB, empireA, NULL, NULL);
        movecar(washA, NULL, NULL, NULL);
        movecar(washB, washA, NULL, NULL);
        movecar(empireA, empireB, NULL, NULL);
// LIGHT HERE
        movecar(empireB, empireC, weyC, NULL);
        movecar(empireC, NULL, NULL, NULL);
// LIGHT HERE
        movecar(dorA, weyA, dorB, NULL);
        movecar(dorB, dorC, westA, NULL);
        movecar(dorC, NULL, NULL, NULL);
        movecar(eddy, weyA, NULL, NULL);
        movecar(mat, westB, NULL, NULL);


// UPDATE CAR COUNT
        totalBefore = totalCars;
        totalCars = pineA->count + pineB->count + weyA->count + weyC->count + richmond->count +
weyB->count + westA->count + westB->count + mat->count + eddy->count + empireA->count +
empireB->count + empireC->count + dorA->count + dorB->count + dorC->count + washA->count +
washB->count;
        if (totalCars != totalBefore)
        {
           carleft++;
        }
        else
        {
           carleft = carleft;
        }
        display(totalCars, carleft, pineA, pineB, richmond, weyA, weyB, weyC, eddy, mat, westA, westB,
washA, washB, dorA, dorB, dorC, empireA, empireB, empireC);
        system("pause");
        whilecnt++;
    }
}

void regularTraffic(struct head *pineA,struct head *pineB,struct head *richmond, struct head *weyA,
struct head *weyB, struct head *weyC, struct head *eddy, struct head *mat, struct head *westA,struct
head *westB, struct head *washA, struct head *washB, struct head *dorA, struct head *dorB, struct head
*dorC, struct head *empireA, struct head *empireB, struct head *empireC)
{
    int r,i, time, totalCars=0, totalBefore=0, carleft=0;
   // r = rand()%4;
   // if (r==0)
    {
        r = rand()%4;
        for (i=0;i<r;i++)
        {
           enqueue(pineA);
        }
    }
  //  if (r==1)
    {
        r = rand()%4;
        for (i=0;i<r;i++)
        {
           enqueue(dorA);
        }
    }
  //  if (r==2)
```

```
    {
        r = rand()%4;
        for (i=0;i<r;i++)
        {
            enqueue(empireA);
        }
    }
  //  if (r==3)
    {
        r = rand()%4;
        for (i=0;i<r;i++)
        {
            enqueue(washB);
        }
    }

    printf("System Run Time:     _____ minutes\n\n");
    scanf("%i",&time);

    totalCars = pineA->count + pineB->count + weyA->count + weyC->count + richmond->count + 
weyB->count + westA->count + westB->count + mat->count + eddy->count + empireA->count + 
empireB->count + empireC->count + dorA->count + dorB->count + dorC->count + washA->count + 
washB->count;
    printf("Simulation Start Total Cars: %i\n\n", totalCars);

    for (i=0;i<time*3;i++)
    {
        r=rand()*18;

        if (r==0)
            {movecar(pineA, pineB, richmond, NULL);}
        if (r==1)
            {movecar(pineB, NULL, NULL, NULL);}
        if (r==2)
            {movecar(richmond, weyC, NULL, NULL);}
        if (r==3)
            {movecar(weyA, weyB, dorA, NULL);}
        if (r==4)
            {movecar(weyB, weyA, NULL, NULL);}
        if (r==5)
            {movecar(weyC, empireC, NULL, NULL);}
        if (r==6)
            {movecar(westA, westB, eddy, NULL);}
        if (r==7)
            {movecar(westB, empireA, NULL, NULL);}
        if (r==8)
            {movecar(washA, NULL, NULL, NULL);}
        if (r==9)
            {movecar(washB, washA, NULL, NULL);}
        if (r==10)
            {movecar(empireA, empireB, NULL, NULL);}
        if (r==11)
            {movecar(empireB, empireC, weyC, NULL);}
        if (r==12)
            {movecar(empireC, NULL, NULL, NULL);}
        if (r==13)
            {movecar(dorA, weyA, dorB, NULL);}
        if (r==14)
            {movecar(dorB, dorC, westA, NULL);}
        if (r==15)
            {movecar(dorC, NULL, NULL, NULL);}
        if (r==16)
            {movecar(eddy, weyA, NULL, NULL);}
        if (r==17)
            {movecar(mat, westB, NULL, NULL);}

        totalBefore = totalCars;

        totalCars = pineA->count + pineB->count + weyA->count + weyC->count + richmond->count +
weyB->count + westA->count + westB->count + mat->count + eddy->count + empireA->count +
empireB->count + empireC->count + dorA->count + dorB->count + dorC->count + washA->count +
washB->count;
        if (totalCars != totalBefore)
        {
            carleft++;
        }
        else
        {
            carleft = carleft;
        }
        display(totalCars, carleft, pineA, pineB, richmond, weyA, weyB, weyC, eddy, mat, westA, westB,
washA, washB, dorA, dorB, dorC, empireA, empireB, empireC);
        system("cls");

    }
    system("pause");
```

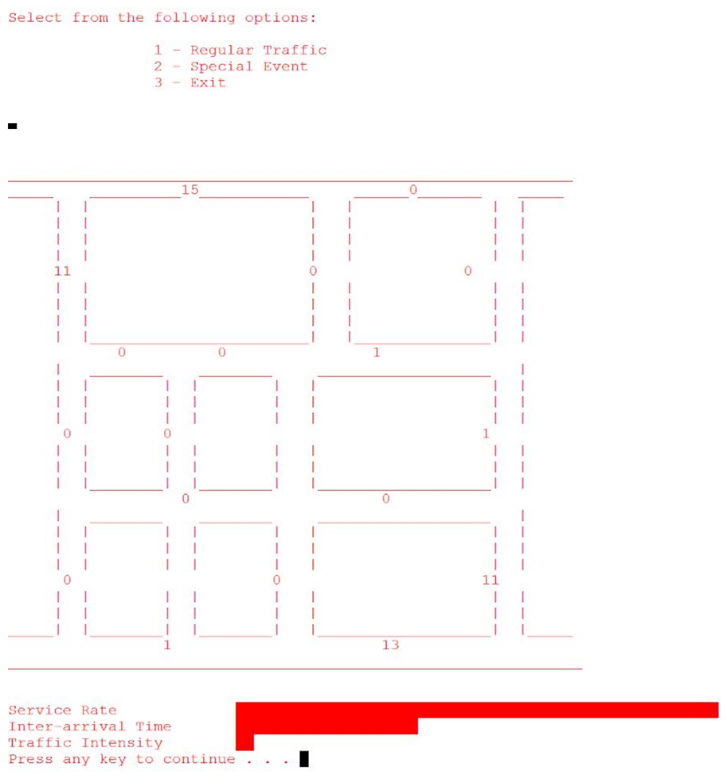
```
}

void display(int tot, int left, struct head *pineA,struct head *pineB,struct head *richmond, struct head
*weyA, struct head *weyB, struct head *weyC, struct head *eddy, struct head *mat, struct head
*westA,struct head *westB, struct head *washA, struct head *washB, struct head *dorA, struct head
*dorB, struct head *dorC, struct head *empireA, struct head *empireB, struct head *empireC)
{
   int ti, i;
   char box = 219;

   printf("______________________________________________________________________\n");
   printf("_____   ___________%i_____________    _______%i________   _____\n",
pineA->count,pineB->count);
   printf("    | |                    |  |            | |    \n");
   printf("    | |                    |  |            | |    \n");
   printf("    | |                    |  |            | |    \n");
   printf("    | |                    |  |            | |    \n");
   printf("    %i                      %i             %i        \n",dorA->count,richmond->count,empireC->count);
   printf("    | |                    |  |            | |    \n");
   printf("    | |                    |  |            | |    \n");
   printf("    | |                    |  |            | |    \n");
   printf("    | |___________________________|  |_________________| |    \n");
   printf("         %i        %i             %i               \n",weyA->count,weyB->count,weyC->count);
   printf("    |  _________   _________    ______________________  |    \n");
   printf("    | |      | |      |  |             | |    \n");
   printf("    | |      | |      |  |             | |    \n");
   printf("    | |      | |      |  |             | |    \n");
   printf("     %i        %i                         %i      \n",dorB->count,eddy->count,empireB->count);
   printf("    | |      | |      |  |             | |    \n");
   printf("    | |      | |      |  |             | |    \n");
   printf("    | |_________| |_________|  |______________________| |    \n");
   printf("              %i                %i               \n",westA->count,westB->count);
   printf("    |  _________   _________    ______________________  |    \n");
   printf("    | |      | |      |  |             | |    \n");
   printf("    | |      | |      |  |             | |    \n");
   printf("    | |      | |      |  |             | |    \n");
   printf("     %i                %i                %i      \n",dorC->count,mat->count, empireA->count);
   printf("    | |      | |      |  |             | |    \n");
   printf("    | |      | |      |  |             | |    \n");
   printf("_____| |_________| |_________|   |______________________|  |_____\n");
   printf("            %i                   %i               \n",washA->count,washB->count);
   printf("______________________________________________________________________\n\n\n");
   printf("Service Rate \t\t ");
```

```
    for(i = 0; i < tot; i++) putchar(box);
    printf("\n");
    printf("Inter-arrival Time \t ");
    for(i = 0; i < left; i++) putchar(box);
    printf("\n");
    if (left != 0)
    {
        ti = tot/left;
    }
    else
    {
        ti=0;
    }
    printf("Traffic Intensity \t ");
    for(i = 0; i < ti; i++) putchar(box);
    printf("\n");
}
```

Testing



```
Select from the following options:

                1 - Regular Traffic
                2 - Special Event
                3 - Exit
```

```
Service Rate
Inter-arrival Time
Traffic Intensity
Press any key to continue . . .
```

After discussing the project and planning it out before we wrote any code, our testing went as follow:

Test 1- The program was on the right track, it accounted for the cars entering/leaving (enqueue/dequeue). It also accounted for street lights and the maximum amount of cars a street can contain before a traffic jam. However, during our first test the code could be more efficient because there were an enqueue and dequeue function for each street in the program.

Test 2- The second test was a working progress. As we worked on a new method to make the enqueue and dequeue function more flexible. We ran into a couple of issues with pointers. In the meantime we kept creating enqueue/dequeue for each street to save some time.

Test 3- The third test proved successful. Our enqueue/dequeue functions are much more flexible. Our program allows only one enqueue/dequeue function for every street. Our hard work proved helpful.

Test 4- The fourth test was us fixing the code that was already written to make it more efficient. We also created more functions when we found repeating-code.

Test5- The fifth test everyone was working according to plan. Our code much less and our hard work payed off. The code was well commented and also contained plenty of functions.

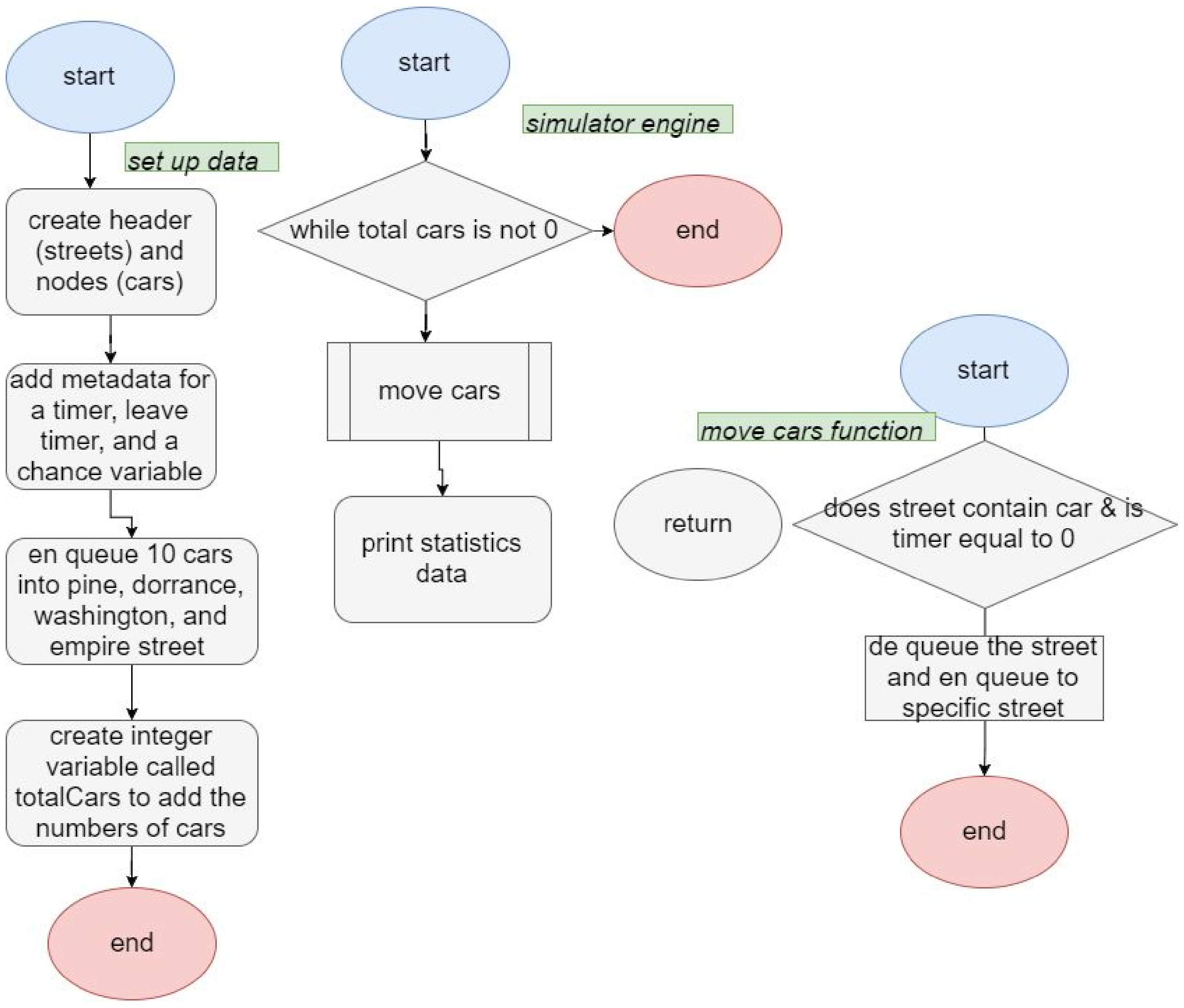
Documentation

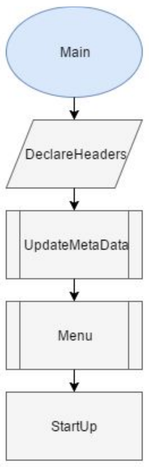
As mentioned in the program description, this project is a real life car simulator created in the C programming language. This program utilizes real data that was collected from multiple streets (Pine, Dorrance, Washington, Eddy, Mathewson) downcity in Providence, Rhode Island. This simulator depicts the traffic intensity that exists in these street. The traffic intensity was possible through the smart use of queues to verify when car has entered/left (queue/dequeue) a street (header).

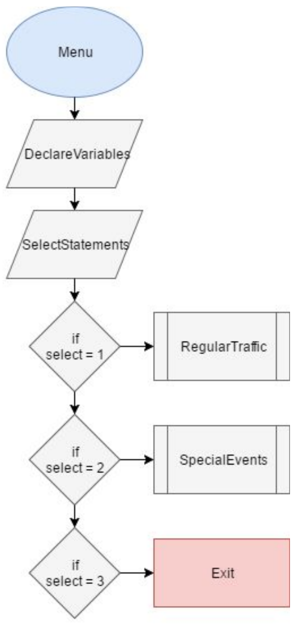
Generalize

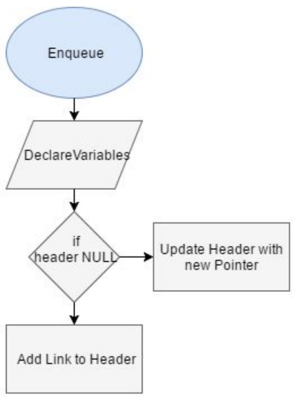
This project is a major advance from our prior snippets. Through the completion of this project we have been able to learn how to use our programming skills and apply it to real life data. It also help to show the growth and knowledge we have obtained over the school year. This street simulator is well done and the traffic intensity is fairly close compared to the actual traffic intensity on these streets. This simulator could be better had we collected data (amount of cars) over longer periods of time. That the way the traffic intensity in the simulator will be even closer. Additionally, this program would be even better if along with the stats there was a visual that showed the cars (nodes) leaving and entering each street (headers) . Lastly, it would be a great improvement if the simulator accounted for car breakdowns, u-turns, car crashes which would make our results even closer to that of real life.

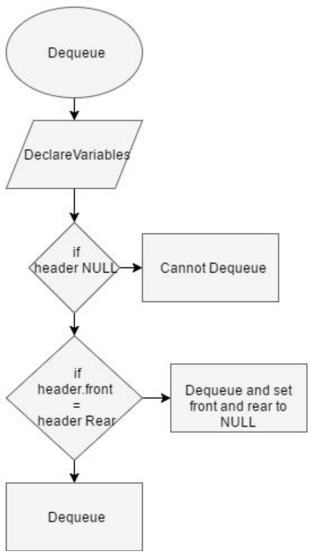
Flow Chart


start
set up data
create header (streets) and nodes (cars)
add metadata for a timer, leave timer, and a chance variable
en queue 10 cars into pine, dorrance, washington, and empire street
create integer variable called totalCars to add the numbers of cars
end
start
simulator engine
while total cars is not 0
end
move cars
print statistics data
start
move cars function
return
does street contain car & is timer equal to 0
de queue the street and en queue to specific street
end


Main
DeclareHeaders
UpdateMetaData
Menu
StartUp


Menu
DeclareVariables
SelectStatements
if select = 1
RegularTraffic
if select = 2
SpecialEvents
if select = 3
Exit


Enqueue
DeclareVariables
if header NULL
Update Header with new Pointer
Add Link to Header


Dequeue
DeclareVariables
if header NULL
Cannot Dequeue
if header.front = header Rear
Dequeue and set front and rear to NULL
Dequeue