Is It Really Your Car…Or Just the Corner?

New Mexico High School
AiS Challenge
Final Report
April 19, 2002

Team #059
Moriarty High School

Team Members:
Jamie O'Dell
Angelica Delgadillo

Team Sponsor:
Paula Avery

Project Mentor:
Dr. John Russell

Executive Summary

Race car drivers spend time determining the math and physics of a course before a race. This has been the complications of racing since the beginning of time. Our basic solution for solving any problems was to apply, “Newton’s Laws of Motion,” using kinematics and dynamic equations, which led us to write two C++ programs. We wanted to determine different driving lines and times. With this we could find the best one, which would give the racer a better benefit in a race. The first one outputs the significant driving lines and times around a ninety degree turn. The second program output’s the significant driving lines and times around a range of radii from 45 to 180 degrees. We accomplished more accurate results than those of Beckman. One major purpose in this project was to determine the relationship of speed and radius. Basically, the width of the track is a major part of what is the safest speed and radius. Therefore, there are not any corresponding radii and speeds. If so, it is very uncommon. This project was well intriguing, yet complicated. There is so much more to racing than just the common turn, straight, turn. Mathematics and physics are a major part and something that is well worth figuring out. This is a problem that could be passed down to anyone whom would like to take it further, which is a future plan.

Introduction

Our program analyzes the physics of a race car corner, enhancing the driver's knowledge of how to drive the course. This could potentially improve the race car time by providing a fast yet safe cornering method. Our project is related to a larger project at UNM School of Engineering. They are trying to find the fastest and safest way to go through a corner and are building the car they will be racing using the physical equations. Students from UNM will use our program to analyze the driving line before they race.

When we started this project we were intrigued by the possibility of writing a program for all race car drivers to use. We chose this particular problem because it concerns a major part of any race car driver's life and a common problem needing to be solved. Using racing simulation is a standard practice in professional race car driving. There is much yet to be discovered in this field and we see an opportunity to contribute to this work. We both have a personal interest in race car driving because we know many people who race competitively.

2. Background

2.1 Problem Definition

The intention of our project is to determine the best way to go through a corner in a race course. By "best," we mean in the least time and at the greatest average speed. We will do this by determining the shape of a driving line through the corner. We will also determine times for multiple lines and compare the results.

This project is significant because it can **only** be approached in two different ways: first by driving experience and skill and the second by the principles of physics.

Understanding of the physics of race car could give a driver a winning edge over the competition.

2.2 Centrifugal or Centripetal Force?

Centrifugal force is the apparent force that throws you to the outside of a turn during cornering. If there is anything loose in the car, it will immediately slide to the right in a left hand turn, and vice versa. Centrifugal force is a fiction, and a consequence of the fact first noticed just over three hundred years ago by Newton that objects tend to continue moving in a straight line unless acted on by an external force (Beckman).

Technically, centrifugal force does not exist. It appears quite real to an object being rotated as illustrated by a child on a merry-go-round. In this situation, the child is not experiencing any real outward force, but he/she must exert a force inwards to keep from flying off the merry-go-round. Since the centrifugal force appears so real, it is often very useful to use as if it were real (Beckman). "The more massive the object, the greater the force." (Newton) We know that this is true because an adult will have a harder time staying on a merry-go-round than a child will.

When you turn the steering wheel, you are trying to get the front tires to push a little sideways on the ground, which then pushes back, according to Newton's third law. When the ground pushes back, it causes a little sideways acceleration. This sideways acceleration is a change in the sideways velocity (Beckman). The forces felt by the driver are centripetal. The term centrifugal means "center fleeing", which refers to the inertial tendency to resist the centripetal force and to continue going straight. If the centripetal force is constant in magnitude, the centrifugal tendency will be constant.

There is no such thing as centrifugal force although it is a convenient fiction for the purpose of some calculations (Beckman).

2.3 Acceleration

The acceleration shown on the graph in diagram C (Appendix E) is usually measured in units of G. The graph is called a GG Diagram. It plots values of G against G. A "G," is a very useful way of measuring acceleration. Acceleration is the change in speed with time, and would typically be measured in units of, for example, meters per second per second. That is how much your speed changed in a given time.

The G manages to remove the matter of weight by comparing measured accelerations with the acceleration due to gravity, which is a reasonably constant value. The result is a number, which has no dimensions; it is unit-less. It is the same regardless of whether you are using metric or imperial units

Acceleration is generally measured using an electro-mechanical device called an Accelerometer. We all have a vertical acceleration of one G acting on us due to the Earth's gravity. A sports car on road tires is unlikely to reach one G in cornering or under braking, though values of around 0.8 to 0.9 G are quite achievable (Beckman).

3. Project Description

3.1 Principles

The physics principles governing cornering in a race car are Newton's 1st and 2nd Laws of Motion. Newton's first law states that every material object continues in its state of rest of uniform motion in a straight line, unless it is compelled to change that state by forces impressed upon it. That is, a car will continue drifting until it is interrupted by an outside force like friction, gravity and in some cases another car.

Newton's second law states that the acceleration of an object is directly proportional to the net force acting on the object, is in the direction of the net force and is inversely proportional to the mass of the object. That is, when a force is applied to a car the change in motion equals the force divided by the mass of the car.

- **Mass** - measure of body's inertia, i.e., resistance to change in state of motion; fundamental measure of amount of matter in body
  - Unit for mass is the *gram*, abbreviation is g
- **Force** - pushes or pulls that change body's state of motion
  - Unit for force is the *dyne*, 1 dyne = 1 g*cm/s^2
- **Velocity** - time rate of change of position (speed) in particular direction; instantaneous velocity (at a single instance of time)
  - Unit for velocity is *centimeter per second*, abbreviation is cm/s
- **Acceleration** - time rate of change of velocity in a particular direction; instantaneous acceleration

3.2 Math Model

For the first program we used the same computational model of Beckman. We defined multiple lines and multiple times using many extensive equations. (c_speed = sqrt(32*radius)*60/88;) : With this equation we were able to define the best cornering speed. We took the square root of the quantity one gee(32) multiplied by any radius, then multiplying the quantity by mph divided by feet per second. We used Beckman's basic calculation of 60mph and 88fps.

Then we determined our braking distance, we used the equation: (b_distance = (100*100-c_speed*c_speed)*88*88/(2*32*3600);) : In this equation we multiplied the quantity of the start speed twice then subtracted the cornering speed squared. Next we multiplied the quantity by feet per second (fps) squared divided by the quantity of two gees by the radii. Later we wanted to determine the straight distance using (s_distance = 13*50-b_distance-100;): With this equation we multiply 13 by 50 subtracting the brake distance. The subtract the starting speed from the brake distance.

Next we want to find the time in the straight so we use (s_time = s_distance/(100*88/60);) : First we divide the straight distance by the quantity of the starting speed multiplied by fps/mph.

After this we wanted to find the time in the braking zone (b_zone = (100-c_speed)*88/(60*32);) : To do this we had to multiply the quantity of the starting speed(100) subtracting the cornering speed. The divide all of that by the quantity mph * amount of g's(32).

To find the time in the corner use the equation (t_corner radius*3.14159265358979/(c_speed*(88/60));) : First multiply the radius by pi. The divide that answer by the quantity cornering speed 8 fps/mph.

Then to find the exit speed from chute is a bit more complicating (e_chute = (sqrt(16*((c_speed*c_speed*88*88/3600)/16+13*50))*60/88-c_speed)*(88/60)/16;):

Just take the square root of the quantity 16 multiplied by the quantity of cornering speed squared multiplied by fps squared divided by 3600. Then divide that quantity by 16 + 13 * 50. After that we just multiplied those two quantities together by mph/fps. Then multiply all of the above by the quantity fps(88)/mph(60) then once you get that number which should be the amount of g's divided by 16.

The exit speed is similar to the time above: (e_speed = sqrt(16*((c_speed*c_speed*88*88/3600)/16+13*50))*60/88;) Just take the square root of the quantity 16 multiplied by the quantity of cornering speed squared multiplied by fps

squared divided by 3600. Then divide that quantity by 16 + 13 * 50. After that we just multiplied those two quantities together by mph/fps.

The total time is the easiest to find by using: ((t_time = s_time + b_zone + t_corner + e_chute;):

Basically just add the time in the straight + the time in the braking zone + time in the corner + and the time in the exit chute.

The second program it became much more complicated because of the fact that it can calculate the corner for any degree turning angle.

ro	outer radius
ri	inner radius
< deg	angle of turn
	accel. Of gravity
g	(32.16)
Ny	lateral g's
Nxa	accelerating g's (.5)
Nxb	braking g's (1.1)
R	Largest radius
SAB	Length of Seg. AB

t	Time
SCD	length of seg. CD
tAB	time in seg. AB
tBC	time in seg. BC
tCD	time in seg. CD
v	Velocity
SBC	length of seg.BC

R = (ro – ri) cos (<deg/2) / 1 – cos(<deg/2) –Largest radius

To get the length of SAB (length of segment AB): SAB = SCD = (R – ri)sin(<deg/2).

To calculate the time in the entry in segment AB use the equations:

VA = VB

SAB = 2(Nxag)(Sab)+VA*VA+VB*VB

If SAB > 0 ,Then:

(VA*VA) = VB*VB+2(nxg)(SAB)

tAA = (VA-VA)/(Nxag) - accelerating

tAB = (VA – VB)(Nxbg) – braking

tAB = tAA + tAB

If SAB < 0 , Then:

tAB = VB – VA / Nxbg

To calculate the time in the turn in segment BC, we used:

VB = sqrt(Nygr) = Vc : With this equation for radius you can use either (r = ro, ri, R)

SBC = r *@

(@ = (<deg/180) * 3.14)

tBC = SBC / VB

To calculate time in the exit in segment BC for ro, ri:

Assume there is no turn coming up which would require braking:

VD = sqrt(2(Nxag)(SCD)+VC*VC)

tCD = VD –VC / Nxag

To calculate the total time, just add all the times together:

t = tAB + tBC + tCD (for ro, ri)

t = tBC (for R)

3.3 Computational Model

We wrote two C++ programs. The first is a replication of Beckman's solution for cornering. In this model, the assumption is that the angle of the corner is 90%. With the help our mentor, John Russell, we added the capability of using any angle to our second program. We verified the accuracy of our second program by running a test case with a cornering angle of 90% and comparing the results to a similar run in our first program.

The results were the same. The third step in this computational model would be to add nested loops representing the radius, the angle, and the width of the corner. This way we could get more data in a single run of the program. We plan on having this program done by the day of the AiS Challenge Expo Awards.

4. Results

We wrote two programs: one we related to Brian Beckman's results and another more flexible. The first code is basically related to a 90 degree angle. The second code was written to process results for any angle. Although, we did think we were inaccurate at the beginning after comparing our results with those of Brian Beckman. We spent time running different numbers through our program. We later found out through our mentor and extensive time spent on the equations that our results were more accurate.

4.1 Knowledge Acquired

In the course of this project we learned more about the physics involved in the dynamics of car racing. Though we all had a basic idea of what racing was prior to this project, we have gained more in-depth knowledge concerning the different methods and stimulations involved in racing.

We also learned some entirely new programming methods and commands. Our project was challenging because of the fact that we needed to use so many equations. We sharpened our previously acquired skills in programming in C++, writing reports, and giving presentations as well. However, we got more out of this project than just new knowledge; we also developed a stronger friendship and experienced working together as a team.

5. Conclusions

Our completed program can be applied to any data set of corners. Although the input section of our program was written to use a numerous amount of radii, the core of the program is entirely general, and the code can be modified to use a different number of radii and speed to determine was is safe and not..

Our program also uses dynamics and kinematics , rather than basic algebra, which means that the user can input different radii and find and accurately safe speed. This is considered to be a good programming technique because it allows the user to define the precise information for the course.

This program and idea was intriguing. We wrote two accurate programs that both out put data. Our problem was solved. We have determined multiple lines and multiple times for a corner of a race course. One program focuses on 90 degree turns and the second focuses on every radii. We too have determined a safe speed

There is only one major limitation to our program: the fact that is only for the corner of the course. This program was written to be extended. The next step is to determine accurate information for a straight. We wrote our program with the knowledge that it is limited to us at the time being.

The success of our program ensures that it will be used and further developed by researchers/students at the engineering school at the University of New Mexico. Our mentor, Dr. Russell, will give our code to his graduate students and have them work with it. They will try to extend our program.

Also, the UNM will use our program to analyze their own data in the future; currently, they are still looking for accurate information for their courses. However, once we are finished we will have successfully helped them with part of their problem.

Racers want to know if there is a correlation between radii and speeds, and our program will help them find out. However, the importance of race simulation is not only for research purposes, but also for accurate personal interests. By knowing the exact radii and speed for the fastest and safest will better benefit the racer. This improves racing accuracy and lets the racer avoid unnecessarily severe accidents.

5.1 Recommendations

Our mentor has suggested that we continue this project next year. We are seriously considering doing so. We would like to extend this project to include the best line and time for racing in a straight. After this has been accomplished we would like our program to be one in which a race car driver could use to find out information for a whole course. It takes time but is something we would love to accomplish. We would like to make the program interactive so that a race driver could readily use the program. One other intriguing idea for our program is graphical and interface output. We have also been offered to join the engineering class at UNM. This project has been so exciting for us and is something we would definitely love to expand on.

5.2 Acknowledgements

We would like to thank everyone who helped us during the course of this project. We could never have finished our project without all their input.

Special gratitude goes to our mentor Dr. John Russell. He was a wonderful mentor throughout the project. He kept contact via e-mail and seven days a week. He donated countless hours of her time to us, both as a mentor and as a friend. Dr. Russell provided us with an opportunity to work on the same project that he has assigned to his graduate students at UNM and he had faith in us that we could do it. He gave us his personal time

for explanations of equations, race car driving, track design, and car design. He also provided research materials, including physic and mathematical papers, and programming tutorials. He explained all aspects of the background information that we didn't understand. He was also an excellent source of programming knowledge; he introduced us with more advanced debugging techniques, and methods for transferring the equations into code. When we had a problem with the code, he was ready to guide us through it. He never just told us the solution; he made us work to figure it out. Dr. Russell works hard for everything that he is involved in and yet still finds time to build and maintain strong relationships with his colleagues, graduate students, and even us. Thank you!

Mrs. Avery is a hard working person. She is involved in many activities and yet still finds time, just like Dr. Russell, to build and maintain a strong relationship with her colleagues, students and especially us! We would like to thank Paula Avery for taking time out of her busy life to sit down and talk things over with us. She too kept into contact via email with us. With her help we gained more knowledge on programming. We also enhanced our presentation skills, with her help. Thank you.

We would also like to thank Brian Beckman, a physicist and a member of the No Bucks Racing Club, who used Newton's principles to develop a set of equations to calculate centripetal force. Without the information that he discovered we would not be able to get anywhere. Thank you.

We would too like to thank Mr. Neil McBeth. Mr. McBeth is a teacher and a team sponsor, and he gave us some interesting presenting skills. Thank you, they were helpful.

Lastly, we would like to thank the people who supported us in our efforts from the beginning: our parents. Our parents… how can we thank them enough? They gave us food, shelter, and transportation, as parents are supposed to, but they took these even farther; they provided meals on the go, they shuttled us between supercomputing meetings. Thanks for believing in us.

References

- Beckman, Brian. “The physics of racing part 4: There is no such Thing a Centrifugal force.” [online] http://www.esbconsult.com.au/ogden/locust/phors/phors04.htm
- Beckman, Brian. “The physics of racing part 5 : Introduction to the racing line.” [online] http://www.esbconsult.com.au/ogden/locust/phors/phors05.htm
- Beckman, Brian. “The physics of racing part 17 : Advanced analysis of the racing line.” [online] http://www.esbconsult.com.au/ogden/locust/phors/phors17.htm
- Beckman, Brian. “The physics of racing part 18 : Advanced racing line, Continued.” [online] http://www.esbconsult.com.au/ogden/locust/phors/phors18.htm
- Hewitt, Paul G. Conceptual Physics Seventh Edition. San Francisco: Harper Collins College Publishers, 1993.

Appendix A
C++ Code Program One

```
// AiS Challenge Program
//Angelica And Jamie

/* */

#include <string.h>
#include <iomanip.h>
#include <conio.h>
#include <iostream.h>
#include <stdlib.h>
#include <math.h>

int radius;             //radius of the corner
//int g = 32;           //gee force
float c_speed;          //cornering speed at 1 g
float b_distance;       //braking distance in ft @ 1g from 100mph
float s_distance;       //straight distance in ft prior to braking
float s_time;           //time in sec in the straight @ 100mph prior to braking
float b_zone;           //time in sec in braking zone
float t_corner;         //time in the corner @ 180 degrees
float e_chute;          //time int the exit chute at 1/2g
float t_time;           //total time in seg
float e_speed;          //exit speed in mph
int x;                  //loop counter
```

```
int main()
{
for (x=50;x<=1000;x+=50)
{
radius=x;
cout <<"radius of the corner "<<radius<<endl;
c_speed = sqrt(32*radius)*60/88;
cout << setprecision(4)<<"c_speed - cornering speed " << c_speed<<endl;
b_distance = (100*100-c_speed*c_speed)*88*88/(2*32*3600);
cout << "b_distance - braking distance " << b_distance<<endl;
s_distance = 13*50-b_distance-100;
cout << setprecision(4)<<"s_distance - straight distance " << s_distance<<endl;
s_time = s_distance/(100*88/60);
cout << setprecision(4)<<"s_time - time in the straight " << s_time<<endl;
b_zone = (100-c_speed)*88/(60*32);
cout <<"b_zone - braking zone " <<b_zone<<endl;
t_corner = radius*3.14159265358979/(c_speed*(88/60));
cout << setprecision(4)<<"t_corner - time in the corner "<<t_corner<<endl;
e_chute = (sqrt(16*((c_speed*c_speed*88*88/3600)/16+13*50))*60/88-
c_speed)*(88/60)/16;
cout << setprecision(4)<<"e_chute - time in the exit chute " <<e_chute<<endl;
t_time =  s_time + b_zone + t_corner + e_chute;
cout << setprecision(4)<<"t_time - exit time " <<t_time<<endl;
e_speed = sqrt(16*((c_speed*c_speed*88*88/3600)/16+13*50))*60/88;
cout << setprecision(4)<<"e_speed - exit speed "<<e_speed<<endl;
system("PAUSE");
}

return 0;
}
```

Appendix B
Code Description

Our program is a complete modeling program in C++ that enables us to try out different computational models in the form of a physic network with variable parameters to see which one works best for our problem. Our code uses the extensive equations obtained from the excel data sets made available by Brian Beckman. This is done by calculating the speed intensities, numerous radius's, into our race stimulation. (See Appendix C: diagram A)

Our completed program can be applied to any data set of race car drivers to classify them into two categories: Dangerous or Safe. Although the input section of our program was written to use a numerous number of radius's and speed using the loop variable, the race stimulation core of the program is entirely general, and the code can be easily modified to use car types.

Our program also uses dynamics and kinematics, rather than simple mathematics, which means that the user can input different speeds, different radius's and get accurate information back. This is considered to be a good programming technique because it allows the user to define the computational environment and it prevents the necessity of compiling the program for each change in numerical radius and speed values.

Appendix C

Program Two

```
//Minitime - isolated curve
//Angelica Delgadillo and Jamie O'Dell

#include <iostream.h>
#include <stdlib.h>
#include<string.h>
#include<iomanip.h>
#include<math.h>

 float g;
 float rcl;
 float w;
 float phi;
 float T;
 float Ro;
 float Ri;
 float ro;
 float ri;
 float ny;
 float nxb;
 float nxa;
 float R;
 float sab;
 float vb;
 float sbc;
 float tbc;
 float tmin;
 float vbi;
 float sbci;
```

```
 float tbci;
 float scdi;
 float vci;
 float vdi;
 float tcdi;
 float va;
 float sapbi;
 float vapi;
 float taapi;
 float tabi;
 float tapbi;
 float ti;
 float vbo;
 float sbco;
 float tbco;
 float scdo;
 float vco;
 float vdo;
 float sapbo;
 float vapo;
 float tapbo;
 float taapo;
 float tabo;
 float to;
 float tcdo;
int main()
{
   g =32.16; //gravitational constant (ft/s^2)

  //curve imput
  rcl = 75;    //radius of corner center line (ft)
```

```
w =30;       //width of course (ft)
phi =90;     //turn angle (degrees)
T =6;        //car track width (ft)

Ro = rcl+.5*w;
Ri =rcl-.5*w;

ro = Ro -.5*T;   //effective outer radius
ri = Ri +.5*T;   //effective inner radius

//race car performance input

ny = 1.10;    //lateral g's
nxb = 1.0;    //braking g's
nxa = 0.5;     //acceleration g's



//calculate largest radius, R, and entry and exit distances, sab
R = (ro - ri*cos(phi*3.14/360))/(1-cos(phi*3.14/360));
sab = (R-ri)*sin(phi*3.145/360);

//calculate time in curve for largest radius R
vb = sqrt (ny*g*R);   //speed in curve
sbc = R*phi*3.145/180;    //curve length
tbc = sbc/vb;          //time in curve
tmin = tbc;

//calculate time in curve with smallest radius, ri

//time in curve
```

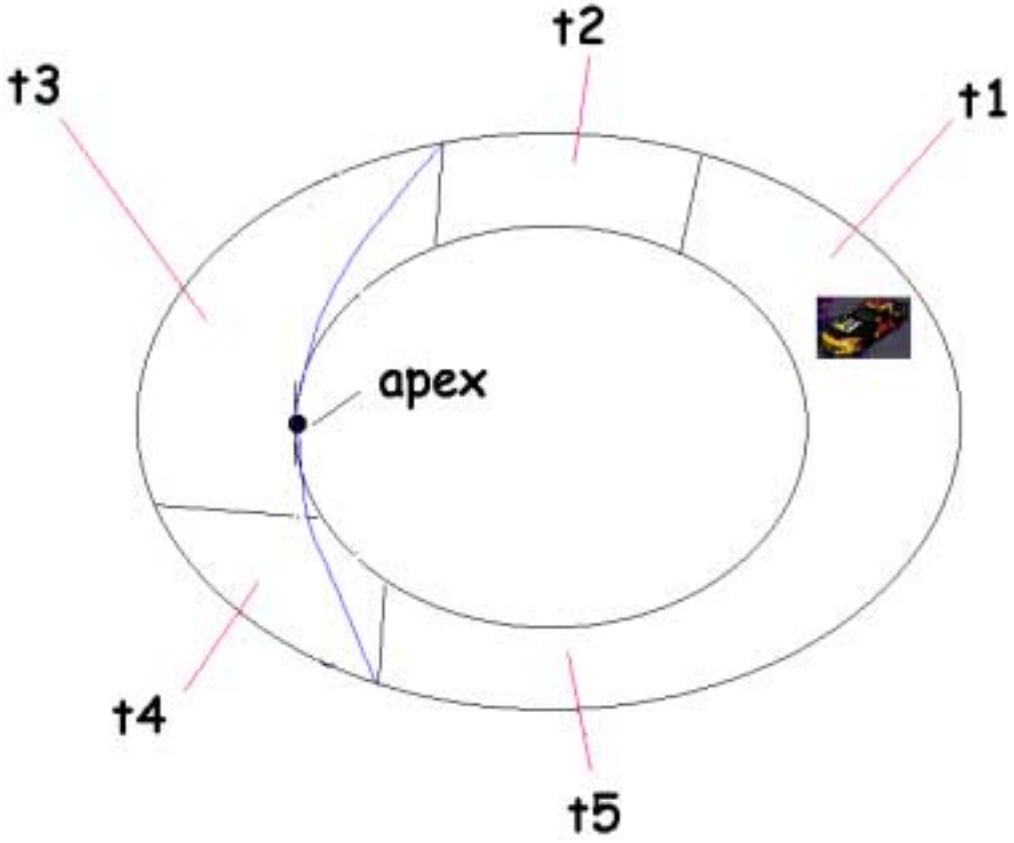
```
vbi = sqrt(ny*g*ri);       //speed in curve
sbci = ri*phi*3.145/180;   //curve length of BC
tbci = sbci/vbi;           //time in curve BC

//time in exit
scdi = sab;            //exit length in same as entry length
vci = vbi;             //speed at C is the same as speed at B
vci = vci*vci;
vdi = sqrt(2*nxa*g*scdi+vci); //speed at D
tcdi =(vdi-vci) /(nxa*g);       //time in exit CD

//time in entrance
va =vb;  ///entrance speed is same as speed for largest radiua curve
va = va*va;
vbi = vbi*vbi;
sapbi = (2.0*nxa*g*sab +va-vbi/2.0*g*(nxa+nxb));
vbi = vbi*vbi;
vapi = sqrt(vbi+2*nxb*g*sapbi);

taapi = ( vapi-va) / (nxa*g);
tapbi = (vapi-vbi)/(nxb*g);

tabi = taapi + tabi;

//total time inner radius

ti = tabi+tbci+tcdi;

//calculate time in curve with largest radius, ro
//time in curve
```

```
vbo = sqrt (ny*g*ro);    //speed in curve
sbco = ro*phi*3.145/180;    //curve length of BC
tbco = sbco/vbo;         //time in curve BC

//time in exit

scdo = sab;              //exit length is same as entry length
vco = vbo;               //speed at C is the same as speed at B
vco = vco*vco;
vdo = sqrt(2*nxa*g*scdo+vco);  //speed at D
tcdo = (vdo -vco)/ (nxa*g);      //time in exit CD

//time in entrance

va = vb;           //entrance speed is same as speed for largest radius curve
va = va*va;
vbo = vbo*vbo;
sapbo = (2*nxa*g*sab+va-vbo)/(2*g*(nxa+nxb));
vbo = vbo*vbo;

vapo = sqrt(vbo+2*nxb*g*sapbo);
taapo = (vapo-va)/(nxa*g);
tapbo = (vapo-vbo)/(nxb*g);

tabo = taapo + tapbo;

//total time outer radius
to = tabo + tbco + tcdo;
  cout << R<<" largest radius of the curve possible"<<endl;
  cout <<rcl<<" radius of the corner center line "<<endl;
  cout <<w<<" width of the course"<<endl;
```

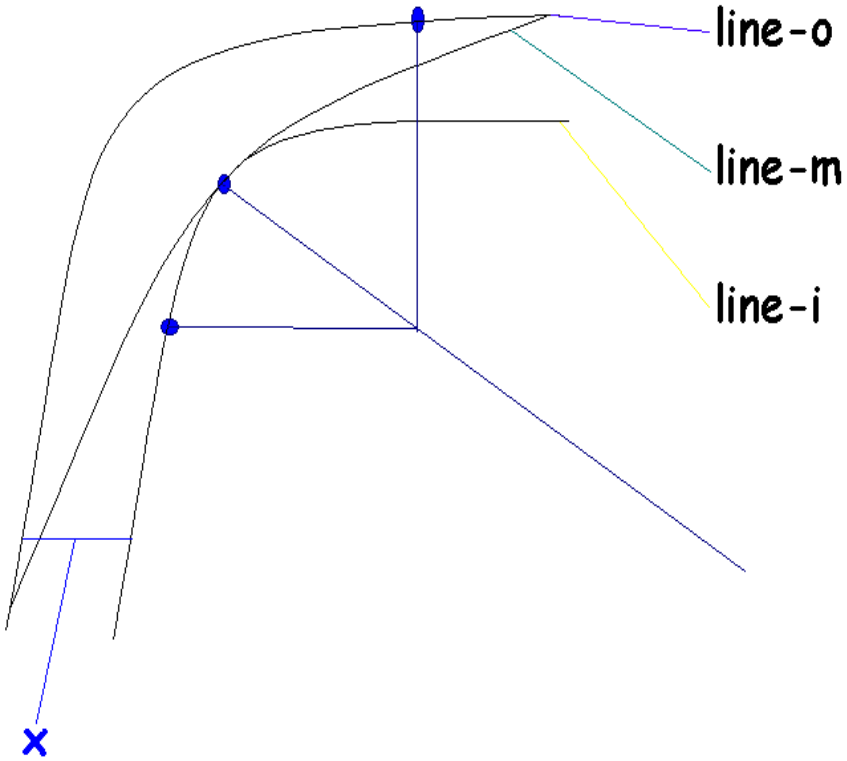
```
    cout <<phi<<" turn angle in degrees"<<endl;
    cout <<T<<" car track width "<<endl;
    cout <<Ro<<" effective outer radius "<<endl;
    cout <<Ri<<" effective inner radius "<<endl;
    //cout
    system("PAUSE");
    return 0;
}
```

Appendix C

Diagram A



Appendix D
Diagram B



Appendix E
Diagram C

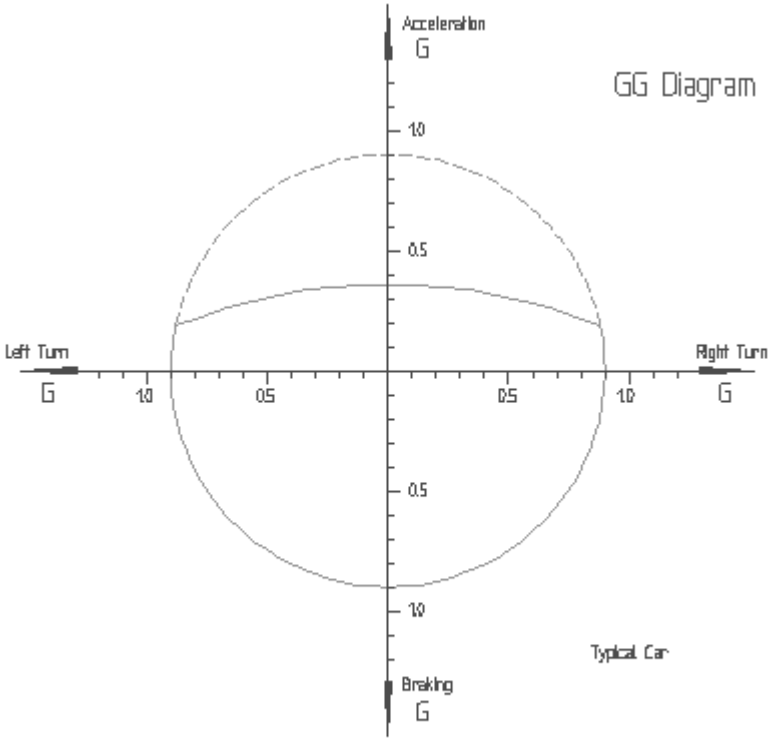


Figure 1

Appendix A

Code

```
//Minitime - isolated curve
#include <iostream.h>
#include <stdlib.h>
#include<string.h>
#include<iomanip.h>
#include<math.h>


    float g;

 float rcl;
 float w;
 float phi;
 float T;
 float Ro;
 float Ri;
 float ro;
 float ri;
 float ny;
 float nxb;
 float nxa;
 float R;
 float sab;
 float vb;
 float sbc;
 float tbc;
 float tmin;
 float vbi;
 float sbci;
 float tbci;
 float scdi;
 float vci;
 float vdi;
 float tcdi;
 float va;
 float sapbi;
 float vapi;
 float taapi;
 float tabi;
 float tapbi;
 float ti;
```

```
 float vbo;
 float sbco;
 float tbco;
 float scdo;
 float vco;
 float vdo;
 float sapbo;
 float vapo;
 float tapbo;
 float taapo;
 float tabo;
 float to;
 float tcdo;
int main()
{
   g =32.16; //gravitational constant (ft/s^2)

  //curve imput
  rcl = 75;    //radius of corner center line (ft)
  w =30;       //width of course (ft)
  phi =30;     //turn angle (degrees)
  T =6;        //car track width (ft)

  Ro = rcl+.5*w;
  Ri =rcl-.5*w;

  ro = Ro -.5*T;   //effective outer radius
  ri = Ri +.5*T;   //effective inner radius

  //race car performance input

  ny = 1.10;    //lateral g's
  nxb = 1.0;    //braking g's
  nxa = 0.5;     //acceleration g's


  //calculate largest radius, R, and entry and exit distances, sab
  R = (ro - ri*cos(phi*3.145/360))/(1-cos(phi*3.145/360));
  sab = (R-ri)*sin(phi*3.145/360);

  //calculate time in curve for largest radius R
  vb = sqrt (ny*g*R);   //speed in curve
  sbc = R*phi*3.145/180;    //curve length
  tbc = sbc/vb;          //time in curve
  tmin = tbc;
```

```
//calculate time in curve with smallest radius, ri

//time in curve

vbi = sqrt(ny*g*ri);       //speed in curve
sbci = ri*phi*3.145/180;   //curve length of BC
tbci = sbci/vbi;           //time in curve BC

//time in exit
scdi = sab;            //exit length in same as entry length
vci = vbi;             //speed at C is the same as speed at B

vdi = sqrt(2*nxa*g*scdi+vci,2); //speed at D
tcdi =(vdi-vci) /(nxa*g);       //time in exit CD

//time in entrance
va =vb;  ///entrance speed is same as speed for largest radiua curve
va = va*va;
vbi = vbi*vbi;
sapbi = (2.0*nxa*g*sab +va-vbi/2.0*g*(nxa+nxb));
vbi = vbi*vbi;
vapi = sqrt(vbi+2*nxb*g*sapbi);

taapi = ( vapi-va) / (nxa*g);
tapbi = (vapi-vbi)/(nxb*g);

tabi = taapi + tabi;

//total time inner radius

ti = tabi+tbci+tcdi;

//calculate time in curve with largest radius, ro
//time in curve

vbo = sqrt (ny*g*ro);    //speed in curve
sbco = ro*phi*3.145/180;    //curve length of BC
tbco = sbco/vbo;         //time in curve BC

//time in exit

                scdo = sab;            //exit length is same as entry length
               vco = vbo;             //speed at C is the same as speed at B

                   vdo = sqrt(2*nxa*g*scdo+vco,2);  //speed at D
                  tcdo = (vdo -vco)/ (nxa*g);      //time in exit CD
```

```
//time in entrance

va = vb;          //entrance speed is same as speed for largest radius curve
va = va*va;
vbo = vbo*vbo;
sapbo = (2*nxa*g*sab+va-vbo)/(2*g*(nxa+nxb));
vbo = vbo*vbo;

vapo = sqrt(vbo+2*nxb*g*sapbo);
taapo = (vapo-va)/(nxa*g);
tapbo = (vapo-vbo)/(nxb*g);

tabo = taapo + tapbo;

//total time outer radius
to = tabo + tbco + tcdo;

system("PAUSE");
return 0;
}
```

```
//Minitime - isolated curve
#include <iostream.h>
#include <stdlib.h>
#include<string.h>
#include<iomanip.h>
#include<math.h>



     float g;

 float rcl;
 float w;
 float phi;
 float T;
 float Ro;
 float Ri;
 float ro;
 float ri;
 float ny;
 float nxb;
 float nxa;
 float R;
 float sab;
 float vb;
 float sbc;
 float tbc;
 float tmin;
 float vbi;
 float sbci;
 float tbci;
 float scdi;
 float vci;
 float vdi;
 float tcdi;
 float va;
 float sapbi;
 float vapi;
 float taapi;
 float tabi;
 float tapbi;
 float ti;
 float vbo;
 float sbco;
 float tbco;
 float scdo;
```

```
 float vco;
 float vdo;
 float sapbo;
 float vapo;
 float tapbo;
 float taapo;
 float tabo;
 float to;
 float tcdo;

int main()
{
   g =32.16; //gravitational constant (ft/s^2)

  //curve imput
  rcl = 75;    //radius of corner center line (ft)
  w = 30;       //width of course (ft)
  phi 90;     //turn angle (degrees)
  T =6;        //car track width (ft)

  Ro = rcl+.5*w;
  Ri = rcl-.5*w;

  ro = Ro -.5*T;   //effective outer radius
  ri = Ri +.5*T;   //effective inner radius

  //race car performance input

  ny = 1.10;    //lateral g's
  nxb = 1.0;    //braking g's
  nxa = 0.5;     //acceleration g's


  //calculate largest radius, R, and entry and exit distances, sab
  R = (ro - ri*cos(phi*3.14/360))/(1-cos(phi*3.14/360));
  sab = (R-ri)*sin(phi*3.145/360);

  //calculate time in curve for largest radius R
  vb = sqrt (ny*g*R);   //speed in curve
  sbc = R*phi*3.145/180;    //curve length
  tbc = sbc/vb;          //time in curve
  tmin = tbc;

  //calculate time in curve with smallest radius, ri

  //time in curve
```

```
vbi = sqrt(ny*g*ri);       //speed in curve
sbci = ri*phi*3.145/180;   //curve length of BC
tbci = sbci/vbi;           //time in curve BC

//time in exit
scdi = sab;            //exit length in same as entry length
vci = vbi;             //speed at C is the same as speed at B
vci = vci*vci;
vdi = sqrt(2*nxa*g*scdi+vci); //speed at D
tcdi =(vdi-vci) /(nxa*g);       //time in exit CD
if (tcdi<0)
tcdi = abs(tcdi);
//time in entrance
va =vb;  ///entrance speed is same as speed for largest radiua curve
va = va*va;
vbi = vbi*vbi;
sapbi = (2.0*nxa*g*sab +va-vbi/2.0*g*(nxa+nxb));
if (sapbi<0)
sapbi=abs(sapbi);
vbi = vbi*vbi;
vapi = sqrt(vbi+2*nxb*g*sapbi);

taapi = ( vapi-va) / (nxa*g);
if (taapi<0)
taapi = abs(taapi);
tapbi = (vapi-vbi)/(nxb*g);
if (tapbi<0)
tapbi=abs(tapbi);
tabi = taapi + tabi;

//total time inner radius

ti = tabi+tbci+tcdi;
if (ti<0)
ti == abs(ti);


//calculate time in curve with largest radius, ro
//time in curve

vbo = sqrt (ny*g*ro);    //speed in curve
sbco = ro*phi*3.145/180;    //curve length of BC
tbco = sbco/vbo;         //time in curve BC

//time in exit
```

```
scdo = sab;            //exit length is same as entry length
vco = vbo;             //speed at C is the same as speed at B
vco = vco*vco;
vdo = sqrt(2*nxa*g*scdo+vco);  //speed at D
tcdo = (vdo -vco)/ (nxa*g);      //time in exit CD
if (tcdo<0)
tcdo = abs(tcdo);
//time in entrance

va = vb;          //entrance speed is same as speed for largest radius curve
va = va*va;
vbo = vbo*vbo;
sapbo = (2*nxa*g*sab+va-vbo)/(2*g*(nxa+nxb));
if (sapbo<0)
sapbo = abs(sapbo);
vbo = vbo*vbo;

vapo = sqrt(vbo+2*nxb*g*sapbo);
if (vapo<0)
vapo = abs(vapo);
taapo = (vapo-va)/(nxa*g);
if (taapo<0)
taapo = abs(taapo);
tapbo = (vapo-vbo)/(nxb*g);
if (tapbo<0)
tapbo = abs(tapbo);

tabo = taapo + tapbo;
if (tabo<0)
tabo = abs(tabo);

//total time outer radius
to = tabo + tbco + tcdo;
if (to<0)
to = abs(to);
  cout << R<<" :largest radius of the curve possible (R) "<<endl;
  cout <<rcl<<" :radius of the corner center line (rcl) "<<endl;
  cout <<w<<" :width of the course (w)"<<endl;
  cout <<phi<<" :turn angle in degrees (phi)"<<endl;
  cout <<T<<" :car track width (T) "<<endl;
  cout <<Ro<<" :effective outer radius (Ro) "<<endl;
  cout <<Ri<<" :effective inner radius (Ri) "<<endl;
  cout <<ro<<" :effective outer radius (ro) "<<endl;
  cout <<ri<<" :effective inner radius (ri) "<<endl;
  cout <<ny<<" :lateral g's (ny) "<<endl;
```

```
    cout <<nxb<<" :braking g's (nxb)"<<endl;
    cout <<nxa<<" :accelerating g's(nxa)"<<endl;
    cout <<sab<<" :time in segment ab (sab) "<<endl;
    cout <<vb<<" :speed in curve (vb) "<<endl;
    cout <<sbc<<" :curve length (sbc) "<<endl;
    system("PAUSE");
    cout <<tbc<<" :time in the curve (tbc)"<<endl;
    cout <<tmin<<" :time in curve (tmin) "<<endl;
    cout <<vbi<<" :speed in curve (vbi) "<<endl;
    cout <<sbci<<" :curve in length BC (sbci) "<<endl;
    cout <<tbci<<" :time in curve BC (tbci) "<<endl;
    cout <<scdi<<" :exit lenth is ame as entry length (scdi) "<<endl;
    cout <<vci<<" :speed at C is the same as speed at B (vci) "<<endl;
    cout <<vdi<<" :speed at D (vdi) "<<endl;
    cout <<tcdi<<" : time in exit CD (tcdi) "<<endl;
    cout <<va<<" :entrance speed is same as speed for largest radius curve (va) "<<endl;
    cout <<sapbi<<" : (sapbi) " <<endl;
    cout <<vapi<<" : (vapi) "<<endl;
    cout <<taapi<<" : (taapi)"<<endl;
    cout <<tapbi<<" : (tapbi)"<<endl;
    cout <<ti<<" :total time in the inner radius (ti) "<<endl;
    system("PAUSE");
    cout <<vbo<<" : speed in the curve (vbo) "<<endl;
    cout <<sbco<<" :curve length of BC (sbco) "<<endl;
    cout <<tbco<<" : (tbco)"<<endl;
    cout <<vco<<" : (vco)"<<endl;
    cout <<vdo<<" : (vdo)"<<endl;
    cout <<tcdo<<" : (tcdo)"<<endl;
    cout <<sapbo<<" : (sapbo)"<<endl;
    cout <<vapo<<" : (vapo)"<<endl;
    cout <<taapo<<" :(taapo)"<<endl;
    cout <<tapbo<<" : (tapbo)"<<endl;
    cout <<tabo<<" : (tabo)"<<endl;
    cout <<to<<" : (to)"<<endl;

    system("PAUSE");
    return 0;
}
```

```
//Minitime - isolated curve
#include <iostream.h>
#include <stdlib.h>
#include<string.h>
#include<iomanip.h>
#include<math.h>



     float g;

 float rcl;
 float w;
 //float phi;
 float T;
 float Ro;
 float Ri;
 float ro;
 float ri;
 float ny;
 float nxb;
 float nxa;
 float R;
 float sab;
 float vb;
 float sbc;
 float tbc;
 float tmin;
 float vbi;
 float sbci;
 float tbci;
 float scdi;
 float vci;
 float vdi;
 float tcdi;
 float va;
 float sapbi;
 float vapi;
 float taapi;
 float tabi;
 float tapbi;
 float ti;
 float vbo;
 float sbco;
 float tbco;
 float scdo;
```

```
 float vco;
 float vdo;
 float sapbo;
 float vapo;
 float tapbo;
 float taapo;
 float tabo;
 float to;
 float tcdo;
 int phi;
int main()
{
   g =32.16; //gravitational constant (ft/s^2)

  //curve imput
  rcl = 75;    //radius of corner center line (ft)
  w = 30;       //width of course (ft)
  //phi 90;     //turn angle (degrees)
  T =6;        //car track width (ft)

  for (phi=45; phi<=180; phi+=22)

  {
  cout<<"Turn angle is = "<<phi<<endl;
  Ro = rcl+.5*w;
  Ri = rcl-.5*w;

  ro = Ro -.5*T;   //effective outer radius
  ri = Ri +.5*T;   //effective inner radius

  //race car performance input

  ny = 1.10;    //lateral g's
  nxb = 1.0;    //braking g's
  nxa = 0.5;     //acceleration g's


  //calculate largest radius, R, and entry and exit distances, sab
  R = (ro - ri*cos(phi*3.14/360))/(1-cos(phi*3.14/360));
  sab = (R-ri)*sin(phi*3.145/360);

  //calculate time in curve for largest radius R
  vb = sqrt (ny*g*R);   //speed in curve
  sbc = R*phi*3.145/180;    //curve length
  tbc = sbc/vb;          //time in curve
  tmin = tbc;
```

```
//calculate time in curve with smallest radius, ri

//time in curve

vbi = sqrt(ny*g*ri);       //speed in curve
sbci = ri*phi*3.145/180;   //curve length of BC
tbci = sbci/vbi;           //time in curve BC

//time in exit
scdi = sab;            //exit length in same as entry length
vci = vbi;             //speed at C is the same as speed at B
vci = vci*vci;
vdi = sqrt(2*nxa*g*scdi+vci); //speed at D
tcdi =(vdi-vci) /(nxa*g);       //time in exit CD
if (tcdi<0)
tcdi = abs(tcdi);
//time in entrance
va =vb;  ///entrance speed is same as speed for largest radiua curve
va = va*va;
vbi = vbi*vbi;
sapbi = (2.0*nxa*g*sab +va-vbi/2.0*g*(nxa+nxb));
if (sapbi<0)
sapbi=abs(sapbi);
vbi = vbi*vbi;
vapi = sqrt(vbi+2*nxb*g*sapbi);

taapi = ( vapi-va) / (nxa*g);
if (taapi<0)
taapi = abs(taapi);
tapbi = (vapi-vbi)/(nxb*g);
if (tapbi<0)
tapbi=abs(tapbi);
tabi = taapi + tabi;

//total time inner radius

ti = tabi+tbci+tcdi;
if (ti<0)
ti == abs(ti);


//calculate time in curve with largest radius, ro
//time in curve

vbo = sqrt (ny*g*ro);    //speed in curve
```

```
sbco = ro*phi*3.145/180;    //curve length of BC
tbco = sbco/vbo;         //time in curve BC

//time in exit

scdo = sab;              //exit length is same as entry length
vco = vbo;               //speed at C is the same as speed at B
vco = vco*vco;
vdo = sqrt(2*nxa*g*scdo+vco);  //speed at D
tcdo = (vdo -vco)/ (nxa*g);      //time in exit CD
if (tcdo<0)
tcdo = abs(tcdo);
//time in entrance

va = vb;           //entrance speed is same as speed for largest radius curve
va = va*va;
vbo = vbo*vbo;
sapbo = (2*nxa*g*sab+va-vbo)/(2*g*(nxa+nxb));
if (sapbo<0)
sapbo = abs(sapbo);
vbo = vbo*vbo;

vapo = sqrt(vbo+2*nxb*g*sapbo);
if (vapo<0)
vapo = abs(vapo);
taapo = (vapo-va)/(nxa*g);
if (taapo<0)
taapo = abs(taapo);
tapbo = (vapo-vbo)/(nxb*g);
if (tapbo<0)
tapbo = abs(tapbo);

tabo = taapo + tapbo;
if (tabo<0)
tabo = abs(tabo);

//total time outer radius
to = tabo + tbco + tcdo;
if (to<0)
to = abs(to);
  cout << R<<" :largest radius of the curve possible (R) "<<endl;
  cout <<rcl<<" :radius of the corner center line (rcl) "<<endl;
  cout <<w<<" :width of the course (w)"<<endl;
  cout <<phi<<" :turn angle in degrees (phi)"<<endl;
  cout <<T<<" :car track width (T) "<<endl;
  cout <<Ro<<" :effective outer radius (Ro) "<<endl;
```

```
    cout <<Ri<<" :effective inner radius (Ri) "<<endl;
    cout <<ro<<" :effective outer radius (ro) "<<endl;
    cout <<ri<<" :effective inner radius (ri) "<<endl;
    cout <<ny<<" :lateral g's (ny) "<<endl;
    cout <<nxb<<" :braking g's (nxb)"<<endl;
    cout <<nxa<<" :accelerating g's(nxa)"<<endl;
    cout <<sab<<" :time in segment ab (sab) "<<endl;
    cout <<vb<<" :speed in curve (vb) "<<endl;
    cout <<sbc<<" :curve length (sbc) "<<endl;
    system("PAUSE");
    cout <<tbc<<" :time in the curve (tbc)"<<endl;
    cout <<tmin<<" :time in curve (tmin) "<<endl;
    cout <<vbi<<" :speed in curve (vbi) "<<endl;
    cout <<sbci<<" :curve in length BC (sbci) "<<endl;
    cout <<tbci<<" :time in curve BC (tbci) "<<endl;
    cout <<scdi<<" :exit lenth is ame as entry length (scdi) "<<endl;
    cout <<vci<<" :speed at C is the same as speed at B (vci) "<<endl;
    cout <<vdi<<" :speed at D (vdi) "<<endl;
    cout <<tcdi<<" : time in exit CD (tcdi) "<<endl;
    cout <<va<<" :entrance speed is same as speed for largest radius curve (va) "<<endl;
    cout <<sapbi<<" : (sapbi) " <<endl;
    cout <<vapi<<" : (vapi) "<<endl;
    cout <<taapi<<" : (taapi)"<<endl;
    cout <<tapbi<<" : (tapbi)"<<endl;
    cout <<ti<<" :total time in the inner radius (ti) "<<endl;
    system("PAUSE");
    cout <<vbo<<" : speed in the curve (vbo) "<<endl;
    cout <<sbco<<" :curve length of BC (sbco) "<<endl;
    cout <<tbco<<" : (tbco)"<<endl;
    cout <<vco<<" : (vco)"<<endl;
    cout <<vdo<<" : (vdo)"<<endl;
    cout <<tcdo<<" : (tcdo)"<<endl;
    cout <<sapbo<<" : (sapbo)"<<endl;
    cout <<vapo<<" : (vapo)"<<endl;
    cout <<taapo<<" :(taapo)"<<endl;
    cout <<tapbo<<" : (tapbo)"<<endl;
    cout <<tabo<<" : (tabo)"<<endl;
    cout <<to<<" : (to)"<<endl;

    system("PAUSE");
    }
     return 0;
}
```