**“Physim”**

**Source Code**

**Source.cpp**

#include"Collisions.h"

/\*Physics Simulator\*/

#include<iostream> //Standard Input Output Header

#include<stdlib.h>

#include<cmath> //Math Library Header for Square Root

#include<time.h> //Time Header for Delay Function

#include<windows.h> //Windows Header contains Declaration and Definitions of Windows API Applications and Functions

#define gravity 9.81 //Constant g for Calculations in Projectile Motion

#define pi 3.14159265 //Constant for Angle Calculations

#include"Collisions.h"

void CriticalAngle(); // Function to Calculate Critical Angle

void centripetalforce(); //functions to calculate centripetal force

void display\_physics\_calculator();

/\*projectile Motion Functions\*/

void projectile(); // function to diplay various calculation opotions in the projectile motion section

void RANGE();// projectile motion functions

void components();

void velocities();

void maxHeight();

/\*co-ordinates conversion functions\*/

void coordinates\_Choices();

void Cartesian\_Cylindrical();

void Cylindrical\_Cartesian();

void xy\_Polar();

void Polar\_xy();

void processingTime();

void DisplayChoices();//displays calculator section

void delay(int);// time delay fucntion

void exitScrn(void); //displays the exit scree when exit() is called

void DisplayMenu();// displaysb main menu

void instruction\_Manual();// displays instruction manual

void main(void)

{

system("cls");//clears console screen

std::cout << "\n\n\n\n\n";//prints the graphical text welcome

std::cout << "\t \_ \_ \_ \_ \n";

std::cout << "\t| | | | | | | |\n";

std::cout << "\t| | | | \_\_\_| | \_\_\_ \_\_\_ \_ \_\_ \_\_\_ \_\_\_ | |\n";

std::cout << "\t| |/\\| |/ \_ | |/ \_\_/ \_ \\| '\_ ` \_ \\ / \_ \\ | |\n";

std::cout << "\t\\ /\\ / \_\_| | (\_| (\_) | | | | | | \_\_/ |\_|\n";

std::cout << "\t \\/ \\/ \\\_\_\_|\_|\\\_\_\_\\\_\_\_/|\_| |\_| |\_|\\\_\_\_| (\_)\n\n\n\n\n\n\n\n";

delay(500);

std::cout << "Loading...";//the following code displays an animation of a loading bar

delay(100);

display\_physics\_calculator();

delay(30);

DisplayMenu();

system("pause");

}

void delay(int a)//time delay function

{

time\_t t1, t2; //two time variables declared

t1 = time(NULL);

t2 = time(NULL);

while (((t2 - t1) \* 1000) < a)//loop until time difference = argument

{

t2 = time(NULL);

}

}

void DisplayChoices() // function displays choices of the calculator function

{

system("cls");//clears screen

int choice;

/\*prompts anf inputs\*/

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SIMULATOR SECTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n";

std::cout << "\n\*\*\*Here you can carry out different calculations regarding Applied Physics Problems\*\*\*\n\n";

std::cout << "Please choose the simulator section related to the problem you entered:\n\n";

std::cout << "1- ";

std::cout << "Centripetal Force\n\n";

std::cout << "2- ";

std::cout << "Projectile Motion\n\n";

std::cout << "3- ";

std::cout << "Cartesian and Cylindrical Co-ordinates\n\n";

std::cout << "4- ";

std::cout << "Elastic Collision\n\n";

std::cout << "0- ";

std::cout << "Go back to Main Menu\n\n";

std::cout << "Please enter your choice: \n(You may also press";

std::cout << " -1";

std::cout << " to exit.)\n";

std::cin >> choice;

for (;;) {

int t;

while ((t = getchar()) != '\n');//exception handling for character input

switch (choice)

{

case 1: centripetalforce();

break;

case 2: projectile();

break;

case 3: coordinates\_Choices();

break;

case 4: ElasticCollision();

choice = 0;

break;

case 0: DisplayMenu();

break;

case -1:exitScrn();

break;

default:

system("cls");

std::cout << "Invalid Choice\n\n";

delay(1000);

DisplayChoices();//receursively calls itself in case of invalid choice

break;

}

}

}

void exitScrn()

{

system("CLS");

std::cout << " \\ / \\ / " << std::endl;

std::cout << " \\ / \\ / " << std::endl;

std::cout << " \\ / \\ / " << std::endl;

std::cout << " \\/ \\/ " << std::endl;

std::cout << " /\\ /\\ " << std::endl;

std::cout << " / \\ / \\ " << std::endl;

std::cout << " / \\ / \\ " << std::endl;

std::cout << " / \\ / \\ " << std::endl << std::endl << std::endl << std::endl << std::endl << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl << std::endl << std::endl;

delay(1000);

system("CLS");//closure screen animation

exit(0);

}

void DisplayMenu()

{

system("cls");

delay(1000);

int choice;

std::cout << "----------------MENU--------------------\n\n\n\n";

/\*prompts and inputs\*/

std::cout << "1- ";

std::cout << "Simulator\n\n\n";

std::cout << "2- ";

std::cout << "Instructions Manual\n\n\n";

std::cout << "Please enter your choice:\n(You may also press";

std::cout << " -1";

std::cout << " to exit.)\n";

std::cin >> choice;

for (;;) {

int t;

while ((t = getchar()) != '\n');//exception handling

{

system("cls");

std::cout << "Invalid choice, please enter a number as input:\n";

}

switch (choice)

{

case 1: DisplayChoices();

break;

case 2: instruction\_Manual();

break;

case -1:exitScrn();

break;

default:

system("cls");

std::cout << "Invalid Choice\n\n";

delay(2000);

DisplayMenu();//recursively calls itself

break;

}

}

}

void instruction\_Manual()

{

system("cls");//clears screen

/\*displays various reules with coloured keywords\*/

std::cout << ("---------------------------INSTRUCTIONS AND GUIDANCE---------------------------\n\n\n");

std::cout << ("The following are some guidelines to familiarize you with the use of the application: \n\n\n");

std::cout << ("1- On most of the simulator sections you can return to previous by only pressing");

std::cout << (" 0");

std::cout << (".\nSome screens will also give you the option to exit the simulator by pressing");

std::cout << (" -1");

std::cout << (".\n\n");

std::cout << ("2- Please avoid using characters in calculations. The program carries out exceptional character\nhandling in menus, however no guarantee can be given about simulations.\n\n\n");

std::cout << ("Enter any number to return back to the Main Menu\n");

int t;

while ((t = getchar()) != '\n')

{

system("cls");

std::cout << ("Warning!\nTry using character inputs.");

DisplayMenu();//displays menu again

}

std::cin >> t;

DisplayMenu();

}

/\*co-ordinates conversion functions\*/

void coordinates\_Choices();

void Cartesian\_Cylindrical();

void Cylindrical\_Cartesian();

void xy\_Polar();

void Polar\_xy();

void processingTime();

void coordinates\_Choices()

{

system("cls");

int choice;

std::cout << ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Cartesian and Cylindrical Co-ordinates\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

std::cout << ("\*\*\*In this section you will be able to convert cartesian co-ordinate to cylindrical co-ordinates and vice versa\*\*\*\n\n");

std::cout << ("\n\nThere are a variety of co-ordinate system coversions you can carry out in this section, please make a choice\nby inputing the number relevant to the choices listed below:\n\n");

std::cout << ("1- ");

std::cout << ("Cartesian to Cylindrical Conversions\n\n");

std::cout << ("2- ");

std::cout << ("Cylindrical to Cartesian Conversians\n\n");

std::cout << ("3- ");

std::cout << ("x-y co-ordinatses to Polar Conversion\n\n");

std::cout << ("4- ");

std::cout << ("Polar to x-y Conversion\n\n");

std::cout << ("0- ");

std::cout << ("Go back to the Calculator section\n\n");

std::cout << ("(Press");

std::cout << (" -1");

std::cout << (" to exit.)\n\n");

std::cout << ("Your choice is: ");

std::cin >> choice;

while (choice != -1)

{

int t;

while ((t = getchar()) != '\n');

switch (choice)

{

case 1: Cartesian\_Cylindrical();

break;

case 2: Cylindrical\_Cartesian();

break;

case 3: xy\_Polar();

break;

case 4: Polar\_xy();

break;

case 0: DisplayChoices();

break;

case -1: exitScrn();

break;

default:

system("cls");

std::cout << ("Invalid Choice\n\n");

delay(2000);

coordinates\_Choices();

break;

}

delay(3000);

system("cls");

coordinates\_Choices();

}

}

void Cartesian\_Cylindrical()

{

long double x, y, z, r, angle;

int angleChoice;

int a;

system("cls");

std::cout << ("Which angular units would you like to use for your calculations:\n");

std::cout << ("1- ");

std::cout << ("Radians\n");

std::cout << ("2- ");

std::cout << ("Degrees\n");

std::cout << ("Please make a choice: ");

std::cin >> angleChoice;

while (angleChoice != 1 && angleChoice != 2)

{

int t;

while ((t = getchar()) != '\n');

system("cls");

std::cout << ("Your previous choice was invalid.\nPlease make a valid choice.\n\n");

std::cout << ("Which angular units would you like to use for your calculations:\n1: Radians\n2: Degrees\nPlease enter your choice:");

std::cin >> angleChoice;

}

system("cls");

std::cout << ("please enter the x, y and z co-ordinates respectively:\n");

std::cout << ("\n\n\nx: ");

std::cin >> x;

std::cout << ("\ny: ");

std::cin >> y;

std::cout << ("\nz: ");

std::cin >> z;

std::cout << "\n\nThe co-ordinates you have provided are\n";

std::cout << x, y, z;

r = sqrt((x \* x + y \* y));

if ((x < 0 && y < 0) || (x < 0 && y>0))

{

angle = -(atan(x / y));

}

else

{

angle = atan(x / y);

}

if (x < 0 && y > 0)

{

angle = angle + (pi / 2);

}

if (x < 0 && y < 0)

{

angle = angle - (pi / 2);

}

if (angleChoice == 2)

{

angle = angle \* (180 / pi);

}

processingTime();

std::cout << "r = ";

std::cout << r;

std::cout << "angle = ";

std::cout << angle;

std::cout << "z = ";

std::cout << z;

std::cout << "The cylindrical co-ordinate equivalent for your input is:\n\n";

std::cout << r << angle << z;

std::cout << "\n\nPress any key to carry out another calcuation:\n";

std::cout << "Press";

std::cout << (" -1");

std::cout << (" to exit.)\n\n");

std::cin >> a;

int t;

while ((t = getchar()) != '\n');

if (a == -1)

{

exitScrn();

}

}

void Cylindrical\_Cartesian()

{

long double x, y, z, r, angle;

int angleChoice;

int a;

system("cls");

std::cout << ("Which angular units would you like to use for your calculations:\n");

std::cout << ("1- ");

std::cout << ("Radians\n");

std::cout << ("2- ");

std::cout << ("Degrees\n");

std::cout << ("Please make a choice: ");

std::cin >> angleChoice;

while (angleChoice != 1 && angleChoice != 2)

{

int t;

while ((t = getchar()) != '\n');

system("cls");

delay(2000);

std::cout << ("Your previous choice was invalid.\nPlease make a valid choice.\n\n");

std::cout << ("Which angular units would you like to use for your calculations:\n1: Radians\n2: Degrees\nPlease enter your choice:");

std::cin >> angleChoice;

}

std::cout << ("please enter the r, angle and z cylindrical co-ordinates respectively:\n");

std::cout << ("\n\n\nr: ");

std::cin >> r;

std::cout << ("\nangle: ");

std::cin >> angle;

std::cout << ("\nz: ");

std::cin >> z;

if (angleChoice == 2)

{

angle = angle \* (pi / 180);

}

if ((angle < 0 && angle >(-(pi / 2))) || (angle > (-(pi)) / 2) && angle < (-(pi)))

{

y = -(r \* cos(angle));

x = -r \* sin(angle);

}

else

{

y = r \* cos(angle);

x = r \* sin(angle);

}

processingTime();

system("cls");

std::cout << ("x = ");

std::cout << x;

std::cout << ("y = ");

std::cout << ("%Lf\n", y);

std::cout << ("z = ");

std::cout << ("The cartesian co-ordinate equivalent for your input is:\n\n");

std::cout << x << y << z;

std::cout << ("\n\nPress any key to carry out another calcuation:\n");

std::cout << ("(Press");

std::cout << (" -1");

std::cout << (" to exit.)\n\n");

std::cin >> a;

int t;

while ((t = getchar()) != '\n');

if (a == -1)

{

exitScrn();

}

}

void xy\_Polar()

{

long double x, y, r, angle;

int angleChoice;

int a;

system("cls");

std::cout << ("Which angular units would you like to use for your calculations:\n");

std::cout << ("1- ");

std::cout << ("Radians\n");

std::cout << ("2- ");

std::cout << ("Degrees\n");

std::cout << ("Please make a choice: ");

std::cin >> angleChoice;

while (angleChoice != 1 && angleChoice != 2)

{

int t;

while ((t = getchar()) != '\n');

system("cls");

std::cout << ("Your previous choice was invalid.\nPlease make a valid choice.\n\n");

std::cout << ("Which angular units would you like to use for your calculations:\n1: Radians\n2: Degrees\nPlease enter your choice:");

std::cin >> angleChoice;

}

system("cls");

std::cout << ("please enter the x and y co-ordinates respectively:\n");

std::cout << ("\n\n\nx: ");

std::cin >> x;

std::cout << ("\ny: ");

std::cin >> y;

std::cout << "\n\nThe co-ordinates you have provided are (" << x << ", " << y << ")" << std::endl;

r = sqrt((x \* x + y \* y));

if ((x < 0 && y < 0) || (x < 0 && y>0))

{

angle = -(atan(x / y));

}

else

{

angle = atan(x / y);

}

if (x < 0 && y > 0)

{

angle = angle + (pi / 2);

}

if (x < 0 && y < 0)

{

angle = angle - (pi / 2);

}

if (angleChoice == 2)

{

angle = angle \* (180 / pi);

}

processingTime();

std::cout << ("r = ");

std::cout << r;

std::cout << ("angle = ");

std::cout << angle;

std::cout << ("The cylindrical co-ordinate equivalent for your input is:\n\n");

std::cout << r << angle;

std::cout << ("\n\nPress any key to carry out another calcuation:\n");

std::cout << ("(Press");

std::cout << " -1";

std::cout << (" to exit.)\n\n");

std::cin >> a;

int t;

while ((t = getchar()) != '\n');

if (a == -1)

{

exitScrn();

}

}

void Polar\_xy()

{

long double x, y, r, angle;

int angleChoice;

int a;

system("cls");

std::cout << ("Which angular units would you like to use for your calculations:\n");

std::cout << ("1- ");

std::cout << ("Radians\n");

std::cout << ("2- ");

std::cout << ("Degrees\n");

std::cout << ("Please make a choice: ");

std::cin >> angleChoice;

while (angleChoice != 1 && angleChoice != 2)

{

int t;

while ((t = getchar()) != '\n');

system("cls");

delay(2000);

std::cout << ("Your previous choice was invalid.\nPlease make a valid choice.\n\n");

std::cout << ("Which angular units would you like to use for your calculations:\n1: Radians\n2: Degrees\nPlease enter your choice:");

std::cin >> angleChoice;

}

std::cout << ("please enter the r and angle co-ordinates respectively:\n");

std::cout << ("\n\n\nr: ");

std::cin >> r;

std::cout << ("\nangle: ");

std::cin >> angle;

if (angleChoice == 2)

{

angle = angle \* (pi / 180);

}

if ((angle < 0 && angle >(-(pi / 2))) || (angle > (-(pi)) / 2) && angle < (-(pi)))

{

y = -(r \* cos(angle));

x = -r \* sin(angle);

}

else

{

y = r \* cos(angle);

x = r \* sin(angle);

}

processingTime();

system("cls");

std::cout << ("x = ");

std::cout << x;

std::cout << ("y = ");

std::cout << y;

std::cout << x << y;

std::cout << ("\n\nPress any key to carry out another calcuation:\n");

std::cout << ("(Press");

std::cout << (" -1");

std::cout << (" to exit.)\n\n");

std::cin >> a;

int t;

while ((t = getchar()) != '\n');

if (a == -1)

{

exitScrn();

}

}

void processingTime()

{

system("cls");

std::cout << ("processing");

delay(500);

std::cout << (".");

delay(300);

std::cout << (".");

delay(300);

std::cout << (".");

system("cls");

}

void velocities()

{

system("cls");

int a;

double velocity, Angle, time;

int option;

std::cout << ("\nEnter Launch Velocity:");

std::cin >> velocity;

std::cout << ("\nEnter Launch Angle:");

std::cin >> Angle;

while (Angle < 0 || Angle>360)

{

system("cls");

std::cout << ("\nERROR! Kindly enter angle between 0 and 360");

std::cin >> Angle;

system("cls");

}

std::cout << ("\nEnter time at which velocity is to be calculated in a trajectory:");

std::cin >> time;

std::cout << ("\nIF YOU WANT TO CALCULATE VELOCITY WHEN OBJECT IS TRAVELLING UP PRESS '1'\nIF YOU WANT TIMEWHEN OBJECT IS TRAVELLING DOWN PRESS '2'\n");

std::cin >> option;

double verticalVelocity = 0;

label:

if (option == 1)

{

verticalVelocity = (velocity\*sin((pi / 180)\*Angle)) - (gravity\*time);

}

else if (option == 2)

{

verticalVelocity = (velocity\*sin((pi / 180)\*Angle)) + (gravity\*time);

}

if (verticalVelocity == 0)

{

std::cout << ("\nARGUEMENT INVALID!!");

std::cout << ("\nENTER A VALID ARGUEMENT(1 or 2):");

std::cin >> option;

goto label;

}

double horizontalVelocity = (velocity\*cos((pi / 180)\*Angle));

double resultant = sqrt((verticalVelocity\*verticalVelocity) + (horizontalVelocity\*horizontalVelocity));

system("cls");

std::cout << ("\n Vertical velocity at that time is %.2lf m/s\n Horizontal velocity retains a constant value of %.2lf m/s throughout the journey.\n Hence the resultant velocity is %.2lf m/s\n", verticalVelocity, horizontalVelocity, resultant);

std::cout << ("\n\nInput any number to carry out another calculation: ");

int t;

while ((t = getchar()) != '\n');

std::cin >> a;

projectile();

}

void RANGE()

{

system("cls");

long double velocity, Angle;

int angleChoice;

int a;

std::cout << ("Which angular units would you like to use for your calculations:\n");

std::cout << ("1- ");

std::cout << ("Radians\n");

std::cout << ("2- ");

std::cout << ("Degrees\n");

std::cout << ("Please make a choice: ");

std::cin >> angleChoice;

while (angleChoice != 1 && angleChoice != 2)

{

system("cls");

delay(2000);

int t;

while ((t = getchar()) != '\n');

std::cout << ("Your previous choice was invalid.\nPlease make a valid choice.\n\n");

std::cout << ("Which angular units would you like to use for your calculations:\n1: Radians\n2: Degrees\nPlease enter your choice:");

std::cin >> angleChoice;

}

system("cls");

std::cout << "\nEnter initial velocity of object:";

std::cin >> velocity;

std::cout << ("\nEnter Launch Angle:");

std::cin >> Angle;

delay(100);

system("cls");

if (angleChoice == 2)

{

while (Angle < 0 || Angle>360)

{

std::cout << ("\nERROR! Kindly enter angle between 0 and 360:");

std::cin >> Angle;

}

{

double range = ((velocity\*velocity)\*(sin(2 \* (pi / 180)\*Angle))) / (2 \* gravity);

processingTime();

std::cout << ("\nThe range of the projectile is ");

std::cout << range;

}

}

else

{

while (Angle < 0 || Angle > 6.284)

{

std::cout << ("\nERROR! Kindly enter angle between 0 and 6.284:");

std::cin >> Angle;

}

{

double range = ((velocity\*velocity)\*(sin(2 \* Angle))) / (2 \* gravity);

processingTime();

std::cout << ("\nThe range of the projectile is ");

std::cout << range;

}

}

std::cout << ("\n\nInput any number to carry out another calculation: ");

int t;

while ((t = getchar()) != '\n');

std::cin >> a;

projectile();

}

void components()

{

system("cls");

long double Force, Angle;

int angleChoice;

int a;

std::cout << ("Which angular units would you like to use for your calculations:\n");

std::cout << ("1- ");

std::cout << ("Radians\n");

std::cout << ("2- ");

std::cout << ("Degrees\n");

std::cout << ("Please make a choice: ");

std::cin >> angleChoice;

while (angleChoice != 1 && angleChoice != 2)

{

system("cls");

delay(2000);

int t;

while ((t = getchar()) != '\n');

std::cout << ("Your previous choice was invalid.\nPlease make a valid choice.\n\n");

std::cout << ("Which angular units would you like to use for your calculations:\n1: Radians\n2: Degrees\nPlease enter your choice:");

std::cin >> angleChoice;

}

system("cls");

std::cout << ("Enter the force:");

std::cin >> Force;

std::cout<<"\nEnter the angle of inclination in degrees:";

std::cin >> Angle;

system("cls");

processingTime();

if (angleChoice == 2) {

while (Angle < 0 || Angle>360)

{

std::cout << ("\nERROR! Kindly enter angle between 0 and 360:");

std::cin >> Angle;

}

double verticalComponent = Force \* sin(Angle\*(pi / 180));

double horizontalComponent = Force \* cos(Angle\*(pi / 180));

std::cout << ("\nThe Horizontal component of the Force is ");

std::cout << ("%.2Lf N while Vertical component is %.2Lf N\n", horizontalComponent, verticalComponent);

}

else

{

while (Angle < 0 || Angle>6.284)

{

std::cout << ("\nERROR! Kindly enter angle between 0 and 2pi:");

std::cin >> Angle;

}

double verticalComponent = Force \* sin(Angle);

double horizontalComponent = Force \* cos(Angle);

std::cout << ("\nThe Horizontal component of the Force is ");

std::cout << ("%.2Lf N while Vertical component is %.2Lf N\n", horizontalComponent, verticalComponent);

}

std::cout << ("\n\nInput any number to carry out another calculation: ");

int t;

while ((t = getchar()) != '\n');

std::cin >> a;

projectile();

}

void maxHeight()

{

system("cls");

long double velocity, angle;

int angleChoice, a;

std::cout << ("Which angular units would you like to use for your calculations:\n");

std::cout << ("1- ");

std::cout << ("Radians\n");

std::cout << ("2- ");

std::cout << ("Degrees\n");

std::cout << ("Please make a choice: ");

std::cin >> angleChoice;

system("cls");

while (angleChoice != 1 && angleChoice != 2)

{

system("cls");

delay(2000);

int t;

while ((t = getchar()) != '\n');

std::cout << ("Your previous choice was invalid.\nPlease make a valid choice.\n\n");

std::cout << ("Which angular units would you like to use for your calculations:\n1: Radians\n2: Degrees\nPlease enter your choice:");

std::cin >> angleChoice;

}

std::cout << "\n\nEnter initial velocity:";

std::cin >> velocity;

std::cout << ("\nEnter launch angle:");

std::cin >> angle;

system("cls");

if (angleChoice == 2) {

while (angle < 0 || angle>360)

{

std::cout << ("\nERROR! Kindly enter angle between 0 and 360");

std::cin >> angle;

}

{

double height = ((velocity\*velocity)\*(sin((pi / 180)\*angle)\*sin((pi / 180)\*angle))) / (2 \* gravity);

std::cout << ("\nThe maximum height attained by object is ");

std::cout << height;

}

}

else

{

while (angle < 0 || angle>6.284)

{

std::cout << ("\nERROR! Kindly enter angle between 0 and 6.284");

std::cin >> angle;

}

{

double height = ((velocity\*velocity)\*(sin((pi / 180)\*angle)\*sin(angle))) / (2 \* gravity);

std::cout << ("\nThe maximum height attained by object is ");

std::cout << height;

}

}

std::cout << ("\n\nInput any number to carry out another calculation: ");

int t;

while ((t = getchar()) != '\n');

std::cin >> a;

projectile();

}

void projectile()

{

system("cls");

int choice;

std::cout << ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Projectile Motion\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

std::cout << ("\*\*\*In this section you will be able to calculate variety of quantities relating to projectiles\*\*\*\n\n");

std::cout << ("\n\nThere are a variety of calculations you can carry out in this section, please make a choice\nby inputing the number relevant to the choices listed below:\n\n");

std::cout << ("1- ");

std::cout << ("Range of projectile\n\n");

std::cout << ("2- ");

std::cout << ("Maximum Height of the projectile\n\n");

std::cout << ("3- ");

std::cout << ("The velocity of the projectile at any point of the flight\n\n");

std::cout << ("4- ");

std::cout << ("Components of force and initial velocity\n\n");

std::cout << ("0- ");

std::cout << ("Go back to Calculator section\n\n");

std::cout << ("(Press");

std::cout << (" -1");

std::cout << (" to exit.)\n\n");

std::cout << ("Your choice is: ");

std::cin >> choice;

while (choice != -1)

{

switch (choice)

{

case 1: RANGE();

break;

case 2: maxHeight();

break;

case 3: velocities();

break;

case 4: components();

break;

case 0: DisplayChoices();

break;

case -1: exitScrn();

break;

default:

system("cls");

std::cout << ("Invalid Choice\n\n");

delay(1000);

DisplayChoices();

break;

}

}

}

void display\_physics\_calculator()

{

system("cls");

std::cout<<"\n\n\n\n\n";

std::cout << "\t\_\_\_\_\_\_ \_ \_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_ \_\_ \n";

std::cout << "\t| \_\_\_ | | | \\ \\ / / \_\_\_|\_ \_/| || || |\n";

std::cout << "\t| |\_/ | |\_| |\\ V /\\ `--. | | | | || | |\n";

std::cout << "\t| \_\_/| \_ | \\ / `--. \\ | | | | | |\n";

std::cout << "\t| | | | | | | | /\\\_\_/ /\_| | | | | |\n";

std::cout << "\t\\\_| \\\_| |\_/ \\\_/ \\\_\_\_\_/ \\\_\_\_/ |\_| |\_|\n\n\n\n\n\n\n\n";

delay(100);

}

void CriticalAngle()

{

long double Refractiveindex, Angle;

int AngleChoice, a;

system("");

std::cout << "Please enter the refractive index of the medium:";

std::cin >> Refractiveindex;

while (Refractiveindex < 1)

{

system("CLS");

std::cout << "Error! Refractive index for the medium cannot be less than 1." << std::endl;

std::cout << "Please enter a refractive index greater than or equal to 1:";

std::cin >> Refractiveindex;

}

system("CLS");

std::cout << "Which angular units would you like to use for your calculations:" << std::endl;

std::cout << "1- ";

std::cout << "Radians" << std::endl;

std::cout << "2- ";

std::cout << "Degrees" << std::endl;

std::cout << "Please make a choice: ";

std::cin >> AngleChoice;

system("CLS");

while (AngleChoice != 1 && AngleChoice != 2)

{

system("CLS");

int t;

while ((t = getchar()) != '\n');

delay(2000);

std::cout << "Your previous choice was invalid." << std::endl;

std::cout << "Please make a valid choice." << std::endl << std::endl;

std::cout << "Which angular units would you like to use for your calculations:" << std::endl;

std::cout << "1: Radians" << std::endl;

std::cout << "2: Degrees" << std::endl;

std::cout << "Please enter your choice : ";

std::cin >> AngleChoice;

}

Angle = asin(1 / Refractiveindex);

if (AngleChoice == 2) {

std::cout << "Critical Angle is ";

std::cout << Angle\*(180 / pi);

}

else {

std::cout << "Critical Angle is ";

std::cout << Angle;

}

std::cout << std::endl << std::endl;

std::cout << "Press any button to return to carry out another calculation: " << std::endl;

std::cout << "Press";

std::cout << " 0 ";

std::cout << "to go back to the Calculator Section" << std::endl << std::endl;

int t;

while ((t = getchar()) != '\n');

std::cin >> a;

if (a == 0)

{

DisplayChoices();

}

DisplayChoices();

}

void centripetalforce()

{

system("cls");

int option, a;

long double mass, velocity, radius, f, force, angularvelocity;

std::cout << "Choose which formula for centripetal force you want to use?";

std::cout << "\nOption ";

std::cout << "1:";

std::cout << "\nF = mv ^ 2 / r\nOption ";

std::cout << "2:";

std::cout << "\nF = mrw ^ 2\n\n";

std::cout << "\nEnter 1 or 2" << std::endl;

std::cout << "Press";

std::cout << " 0 ";

std::cout << "to go back to the Calculator Section)\n" << std::endl;

std::cin >> option;

system("cls");

switch (option)

{

case 1:

{

std::cout << "Enter the mass: ";

std::cin >> mass;

std::cout << "Enter the velocity: ";

std::cin >> velocity;

std::cout << "Enter the radius: ";

std::cin >> radius;

f = mass \* velocity\*velocity;

force = f / radius;

}

break;

case 2:

{

std::cout<<"Enter the mass: ";

std::cin >> mass;

std::cout << "Enter the angular velocity: ";

std::cin >> angularvelocity;

std::cout << "Enter the radius: ";

std::cin >> radius;

force = mass \* radius \* angularvelocity\*angularvelocity;

}

break;

case 0: DisplayChoices();

}

processingTime();

std::cout << "Centripetal force is : ";

std::cout << force;

std::cout << "\n\nPress any button to return to carry out another calculation: ";

int t;

while ((t = getchar()) != '\n');

std::cin >> a;

}

**PhysicalQuantities.h**

#ifndef PHYSICALQUANTITIES\_H

#define PHYSICALQUANTITIES\_H

#include<cmath>

#include<iostream>

//velocity

class Velocity

{

protected:

double position[3];

double velocity[3];

double updatedPosition[3];

public:

//constructors

Velocity(){}

Velocity(double x1, double y1, double z1, double x2, double y2, double z2);

Velocity(double x, double y, double z);

//setters

void setPosition(double, double, double);

void setVelocity(double, double, double);

//getters

void getPosition(double arr[]);

void getVelocity(double arr[]);

void getUpdatedPosition(double arr[]);

//updated position

void updatePosition(double);

//magnitude of velocity

inline const double magnitudeVelocity(void);

//Destructor

~Velocity(){}

};

//Acceleration

class Acceleration : public Velocity

{

protected:

double acceleration[3];

public:

//Constructors

Acceleration(double x1, double y1, double z1, double x2, double y2, double z2);

Acceleration(double x, double y, double z);

//Getters

void getAcceleration(double arr[3]);

//Setters

void setAcceleration(double x, double y, double z);

//Update Velocity

void updateVelocity(double);

//Destructor

~Acceleration();

};

#endif

**PhysicalQuantities.cpp**

#include"PhysicalQuantities.h"

//velocity

//Constructors

Velocity::Velocity(double x1, double y1, double z1, double x2, double y2, double z2)

{

position[0] = x1; position[1] = y1; position[2] = z1;

velocity[0] = x2; velocity[1] = y2; velocity[2] = z2;

updatedPosition[0] = x1; updatedPosition[1] = y1; updatedPosition[2] = z1;

}

Velocity::Velocity(double x, double y, double z)

{

position[0] = x; position[1] = y; position[2] = z;

velocity[0] = 0; velocity[1] = 0; velocity[2] = 0;

updatedPosition[0] = x; updatedPosition[1] = y; updatedPosition[2] = z;

}

//setters

void Velocity::setPosition(double x1, double y1, double z1)

{

position[0] = x1; position[1] = y1; position[2] = z1;

}

void Velocity::setVelocity(double x1, double y1, double z1)

{

velocity[0] = x1; velocity[1] = y1; velocity[2] = z1;

}

//getters

void Velocity::getPosition(double arr[])

{

for (int i = 0; i < 3; i++)

{

arr[i] = position[i];

}

}

void Velocity::getVelocity(double arr[])

{

for (int i = 0; i < 3; i++)

{

arr[i] = velocity[i];

}

}

void Velocity::getUpdatedPosition(double arr[])

{

for (int i = 0; i < 3; i++)

{

arr[i] = updatedPosition[i];

}

}

//updated velocity

void Velocity::updatePosition(double t)

{

for (int i = 0; i < 3; i++)

{

updatedPosition[i] = position[i] + velocity[i] \* t; //new position = old position + distance travelled (s = vt)

}

}

//magnitude of velocity

inline const double Velocity::magnitudeVelocity(void)

{

return pow((pow(velocity[0], 2) + pow(velocity[1], 2) + pow(velocity[2], 2)), (1 / 2));

}

//Acceleration

//Constructors

Acceleration::Acceleration(double x, double y, double z) : Velocity(x, y, z)

{

acceleration[0] = 0;

acceleration[1] = 0;

acceleration[2] = 0;

}

Acceleration::Acceleration(double ax, double ay, double az, double vx, double vy, double vz) : Velocity(vx, vy, vz)

{

acceleration[0] = ax;

acceleration[1] = ay;

acceleration[2] = az;

}

//Getters

void Acceleration::getAcceleration(double arr[3])

{

arr[0] = acceleration[0];

arr[1] = acceleration[1];

arr[2] = acceleration[2];

}

//Setters

void Acceleration::setAcceleration(double x, double y, double z)

{

acceleration[0] = x;

acceleration[1] = y;

acceleration[2] = z;

}

//Update Velocity

void Acceleration::updateVelocity(double t)

{

for (int i = 0; i < 3; i++)

{

velocity[i] = velocity[i] + acceleration[i] \* t;

}

}

//Destructor

Acceleration::~Acceleration(){}

**PhysicalObjects.h**

#include"PhysicalQuantities.h"

//Sphere

class Sphere : public Velocity

{

private:

double mass;

public:

Sphere(double m, double x1, double y1, double z1, double x2, double y2, double z2) : mass(m), Velocity(x1, y1, z1, x2, y2, z2) {}

Sphere(double m, double x, double y, double z) : mass(m), Velocity(x, y, z) {}

//static functions can only be called/invoked using scope resolution operator

static double distance(Sphere, Sphere); //static function returns distance between two circles

static double updatedDistance(Sphere, Sphere);

//calculating "radius"

double radius(void);

//getter and setter for mass

void setMass(double);

double getMass(void);

//Destructor

~Sphere(){}

};

**PhysicalObjects.cpp**

#include"PhysicalObjects.h"

//sphere

//"radius"

double Sphere::radius(void)

{

return 100 \* cbrt((mass \* 3) / 1000);

}

double Sphere::distance(Sphere s1, Sphere s2)

{

double distBWcentres;

//d1 is the distance between centeres of two radii

//d1 is calculated by taking difference of magnitudes of two centre vectors

distBWcentres = sqrt(((s2.position[0] - s1.position[0])\*(s2.position[0] - s1.position[0])) + ((s2.position[1] - s1.position[1])\*(s2.position[1] - s1.position[1])) + ((s2.position[2] - s1.position[2])\*(s2.position[2] - s1.position[2])));

return distBWcentres - (s1.radius() + s2.radius());

}

double Sphere::updatedDistance(Sphere s1, Sphere s2)

{

double distBWcentres;

//d1 is the distance between centeres of two radii

//d1 is calculated by taking difference of magnitudes of two centre vectors

distBWcentres = sqrt(((s2.updatedPosition[0] - s1.updatedPosition[0])\*(s2.updatedPosition[0] - s1.updatedPosition[0])) + ((s2.updatedPosition[1] - s1.updatedPosition[1])\*(s2.updatedPosition[1] - s1.updatedPosition[1])) + ((s2.updatedPosition[2] - s1.updatedPosition[2])\*(s2.updatedPosition[2] - s1.updatedPosition[2])));

return distBWcentres - (s1.radius() + s2.radius());

}

// setter and getter for "mass"

void Sphere::setMass(double m)

{

mass = m;

}

double Sphere::getMass(void)

{

return mass;

}

**Collisions.h**

#ifndef COLLISIONS\_H

#define COLLISIONS\_H

#include"PhysicalObjects.h"

#include"PhysicalQuantities.h"

bool checkCollision(Sphere s1, Sphere s2)

{

double dist1, dist2;

dist1 = Sphere::updatedDistance(s1, s2);

while (!(dist1 < 0.5 && dist1 > 0))

{

static double t = 0;

dist1 = Sphere::updatedDistance(s1, s2);

s1.updatePosition(t);

s2.updatePosition(t);

dist2 = Sphere::updatedDistance(s1, s2);

if (dist1 < 0.5 && dist1 > 0)

{

return true;

}

if (dist2 > dist1)

{

return false;

}

t += 0.03;

}

}

int collision(Sphere s1, Sphere s2)

{

bool check = checkCollision(s1, s2);

if (!check)

{

std::cout << "\n\n\tSpheres don't collide!\n\n" << std::endl;

return 0;

}

double dist = Sphere::distance(s1, s2);

while (dist > 0)

{

static double t = 0;

s1.updatePosition(t);

s2.updatePosition(t);

dist = Sphere::updatedDistance(s1, s2);

if (dist < 0.5 && dist > 0)

{

break;

}

t += 0.03;

}

//after collision

double a, b, c, d;

a = (s1.getMass() - s2.getMass()) / (s1.getMass() + s2.getMass());

b = (2 \* (s2.getMass())) / (s1.getMass() + s2.getMass());

c = (2 \* (s1.getMass())) / (s1.getMass() + s2.getMass());

d = (s2.getMass() - s1.getMass()) / (s2.getMass() + s1.getMass());

double V1[3], V2[3];

s1.getVelocity(V1);

s2.getVelocity(V2);

double V1Prime[] = { ((a\*V1[0]) + (b\*V2[0])), ((a\*V1[1]) + (b\*V2[1])), ((a\*V1[2]) + (b\*V2[2])) };

double V2Prime[] = { ((c\*V1[0]) + (d\*V2[0])), ((c\*V1[1]) + (d\*V2[1])), ((c\*V1[2]) + (d\*V2[2])) };

s1.setVelocity(V1Prime[0], V1Prime[1], V1Prime[2]);

s2.setVelocity(V2Prime[0], V2Prime[1], V2Prime[2]);

std::cout << "\n\n\tV1' = " << V1Prime[0] << "i + " << V1Prime[1] << "j + " << V1Prime[2] << "k" << std::endl;

std::cout << "\n\n\tV2' = " << V2Prime[0] << "i + " << V2Prime[1] << "j + " << V2Prime[2] << "k" << std::endl;

int t;

while ((t = getchar()) != '\n')

{collision(s1, s2); }

return 1;

}

void ElasticCollision(void)

{

system("cls");

double m1, m2, v1x, v1y, v1z, v2x, v2y, v2z;

std::cout << "\n Mass" << std::endl;

std::cout << " m1 = ";

std::cin >> m1;

fflush(stdin);

std::cout << " m2 = ";

std::cin >> m2;

fflush(stdin);

std::cout << "\n Velocities" << std::endl;

std::cout << "\n Sphere - 1" << std::endl;

std::cout << " Vx1 = ";

std::cin >> v1x;

fflush(stdin);

std::cout << " Vy1 = ";

std::cin >> v1y;

fflush(stdin);

std::cout << " Vz1 = ";

std::cin >> v1z;

fflush(stdin);

std::cout << "\n Sphere - 2" << std::endl;

std::cout << " Vx2 = ";

std::cin >> v2x;

fflush(stdin);

std::cout << " Vy2 = ";

std::cin >> v2y;

fflush(stdin);

std::cout << " Vz2 = ";

std::cin >> v2z;

fflush(stdin);

Sphere s1(m1, 10, 10, 10, v1x, v1y, v1z), s2(m2, 120, 10, 10, v2x, v2y, v2z);

collision(s1, s2);

system("pause");

}

#endif