Programación para Modelado y Simulación: Sesion: 14, Programación orientada a Objetos y Dinámica Molecular Básica

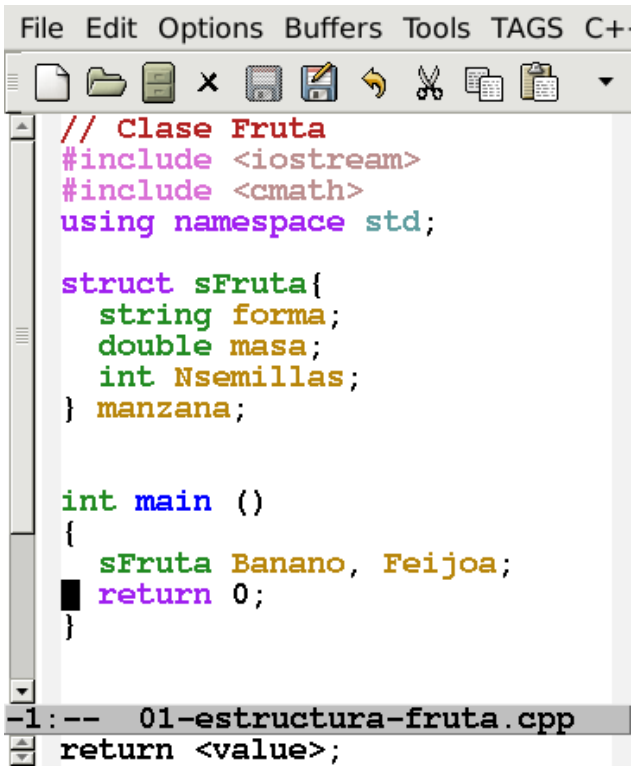
Profesores: Gabriel Villalobos Camargo. ,
(gabriel.villalobosc@utadeo.edu.co)
Camilo Andrés Espejo Pabón. , (camilo.espejo@utadeo.edu.co)

14 de noviembre de 2014

Estructuras de datos

Características

- Son **grupos** de **elementos** agrupados bajo un sólo nombre
- Se declaran:
  `struct NombreDeLaEstructura { TipoMiembro1 NombreMiembro1; TipoMiembro2 NombreMiembro2; TipoMiembro3 NombreMiembro3; }NombreObjeto;`
- Al delcararla se crea un nuevo tipo!



Acceder a Miembros

" . "

Los miembros de los **tipos de estructuras** existen para cada **estructura**. Cada estructura, del tipo sFruta, tendrá forma, masa y Nsemillas.
Se accede a ellos con un punto después del nombre:

File Edit Options Buffers Tools TAGS C++ Senator

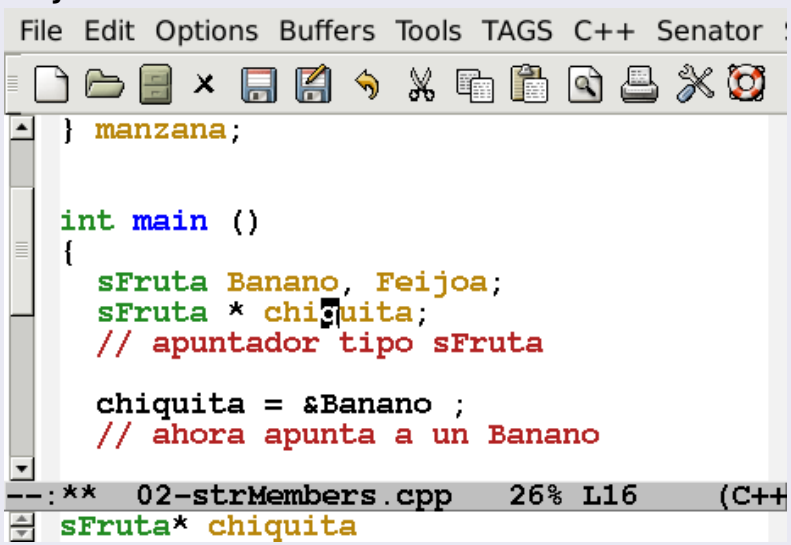
```
// Clase Fruta
#include <iostream>
#include <cmath>
using namespace std;

struct sFruta{
  string forma;
  double masa;
  int Nsemillas;
} manzana;


int main ()
{
  sFruta Banano, Feijoa;
  manzana.Nsemillas = 4;
  manzana.masa = 300;
  manzana.forma = "manzanoidea";

  Banano.Nsemillas = 0;
  Banano.masa=250;
  Banano.forma = "alargado";

  cout << Banano.Nsemillas << endl;
  cout << manzana.forma << endl;

  return 0;
}
```

-1:-- 02-strMembers.cpp Top L26 (C++

return <value>;

0. Pointers

De una estructura se puede hacer un puntero. Ej. chiquita es puntero tipo sFruta. &Banano es la dirección del objeto Banano.

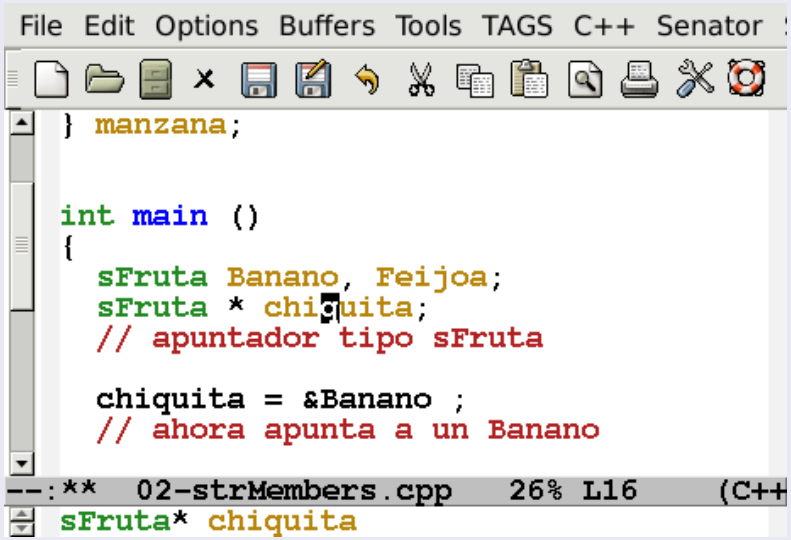


1. Pointers

El asterisco * denota un puntero. Para referirnos a un miembro del objeto apuntado por un puntero se pueden usar dos métodos. Uno es la flecha: `chiquita->masa`, se pude intepretar como “la cantidad masa del objeto al que apunta el puntero chiquita”.
La segunda forma es “dereferenciar” el puntero, usando el paréntesis y el asterisco: `(*chiquita)` es el objeto al que apunta chiquita.
Entonces: `(*chiquita).masa` se puede interpretar como “La masa del banano (que es el objeto al que apunta el puntero chiquita”.
Parece un trabalenguas!
Lo que no funciona (ERROR COMUN) es : *chiquita.masa.

0. Pointers

De una estructura se puede hacer un puntero. Ej. chiquita es puntero tipo sFruta. &Banano es la dirección del objeto Banano.



1. Pointers

El asterisco * denota un puntero. Para referirnos a un miembro del objeto apuntado por un puntero se pueden usar dos métodos. Uno es la flecha: `chiquita->masa`, se pude intepretar como “la cantidad masa del objeto al que apunta el puntero chiquita”.

La segunda forma es “dereferenciar” el puntero, usando el paréntesis y el asterisco: `(*chiquita)` es el objeto al que apunta chiquita.

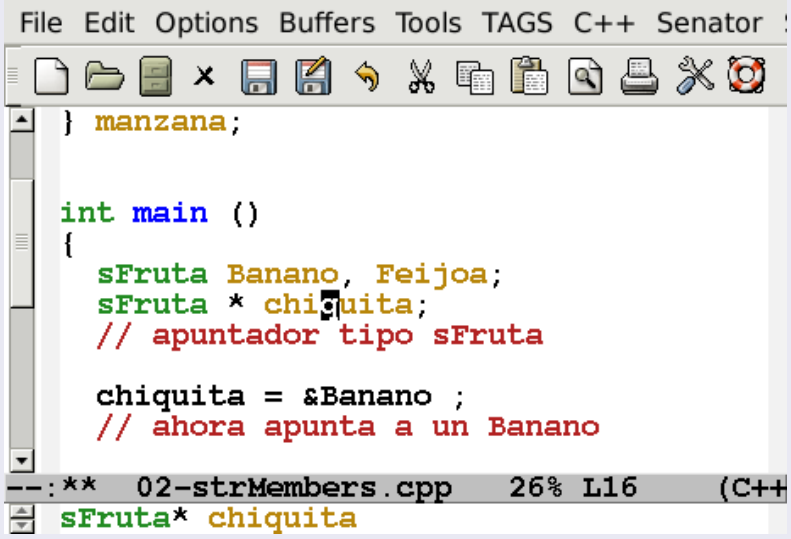
Entonces: `(*chiquita).masa` se puede interpretar como “La masa del banano (que es el objeto al que apunta el puntero chiquita”.

Parece un trabalenguas!

Lo que no funciona (ERROR COMUN) es : *chiquita.masa.

0. Pointers

De una estructura se puede hacer un puntero. Ej. chiquita es puntero tipo sFruta. &Banano es la dirección del objeto Banano.



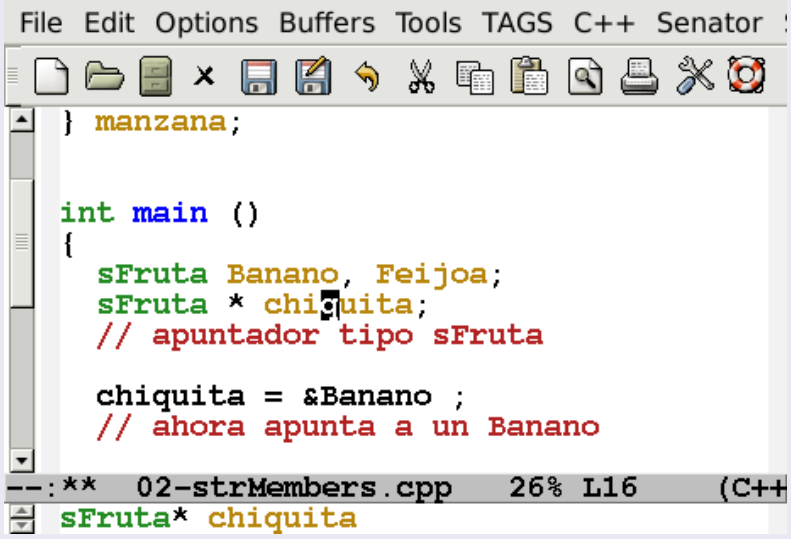
1. Pointers

El asterisco * denota un puntero. Para referirnos a un miembro del objeto apuntado por un puntero se pueden usar dos métodos. Uno es la flecha: `chiquita->masa`, se pude intepretar como “la cantidad masa del objeto al que apunta el puntero chiquita”.
La segunda forma es “dereferenciar” el puntero, usando el paréntesis y el asterisco: `(*chiquita)` es el objeto al que apunta chiquita.
Entonces: `(*chiquita).masa` se puede interpretar como “La masa del banano (que es el objeto al que apunta el puntero chiquita”.
Parece un trabalenguas!
Lo que no funciona (ERROR COMUN) es : *chiquita.masa.

UTADEO
UNIVERSIDAD DE BOGOTÁ JORGE TADEO LOZANO

0. Pointers

De una estructura se puede hacer un puntero. Ej. chiquita es puntero tipo sFruta. &Banano es la dirección del objeto Banano.



1. Pointers

El asterisco * denota un puntero. Para referirnos a un miembro del objeto apuntado por un puntero se pueden usar dos métodos. Uno es la flecha: `chiquita->masa`, se pude intepretar como “la cantidad masa del objeto al que apunta el puntero chiquita”.
La segunda forma es “dereferenciar” el puntero, usando el paréntesis y el asterisco: `(*chiquita)` es el objeto al que apunta chiquita.
Entonces: `(*chiquita).masa` se puede interpretar como “La masa del banano (que es el objeto al que apunta el puntero chiquita”.
Parece un trabalenguas!
Lo que no funciona (ERROR COMUN) es : *chiquita.masa.

CLASES

Características

- Como las estructuras, agrupan tipos de datos.
- Pero también tienen FUNCIONES!

```
File  Edit  Options  Buffers  Tools  TAGS  C++  Senator  SRecoder  H
// Clase Fruta
#include <iostream>
#include <cmath>
using namespace std;

class CFruta{
private:
   string forma;
   double masa;
   int Nsemillas;
public:
   CFruta (string, double, int);
   string digaForma (void){return forma;};
   double digaMasa (void){return masa;};
   double digaNsemillas (void){return Nsemillas;
};
};

CFruta::CFruta (string aa, double xx, int nn){
   forma = aa ; masa = xx ; Nsemillas = nn ; }

int main ()
{
   CFruta Manzana("esférica", 120, 4);
   cout << Manzana.digaForma()<< endl;
   cout << Manzana.digaMasa()<< endl;
   cout << Manzana.digaNsemillas()<< endl;
-1:--  03-clase-fruta.cpp    Top L1      (C++/l Abbre
```

Operadores

Palabra reservada `operator`

Permite redefinir un operador, por ejemplo >, para que se entienda como una función entre los objetos de la clase.

```
// Clase Fruta
#include <iostream>
#include <cmath>
#include <locale>

using namespace std;

class CFruta{
private:
  string forma;
  double masa;
  int Nsemillas;
public:
  CFruta (string, double, int);
  bool operator> (CFruta B){if (masa>B.masa) return true; else return false;};
  bool operator< (CFruta B){if (masa<B.masa) return true; else return false;};
  string digaForma (void){return forma;};
  double digaMasa (void){return masa;};
  double digaNsemillas (void){return Nsemillas;};
};

CFruta::CFruta (string aa, double xx, int nn){
  forma = aa ; masa = xx ; Nsemillas = nn ; }

int main ()
{
  CFruta Manzana("esferica  ", 120, 4);
  CFruta Papaya ("papayoidea", 222, 4);
  if (Manzana>Papaya)
    cout << Manzana.digaForma()<< endl;
  else
    cout << Papaya.digaForma()<< endl;


  return 0;
}
```

Pierre-Simon Laplace

We may regard the present state of the universe as the effect of its past and the cause of its future. An intellect which at a certain moment would know all forces that set nature in motion, and all positions of all items of which nature is composed, if this intellect were also vast enough to submit these data to analysis, it would embrace in a single formula the movements of the greatest bodies of the universe and those of the tiniest atom; for such an intellect nothing would be uncertain and the future just like the past would be present before its eyes.

Pierre Simon Laplace, A phillosophical Essay on Probabilities

Dinámica molecular

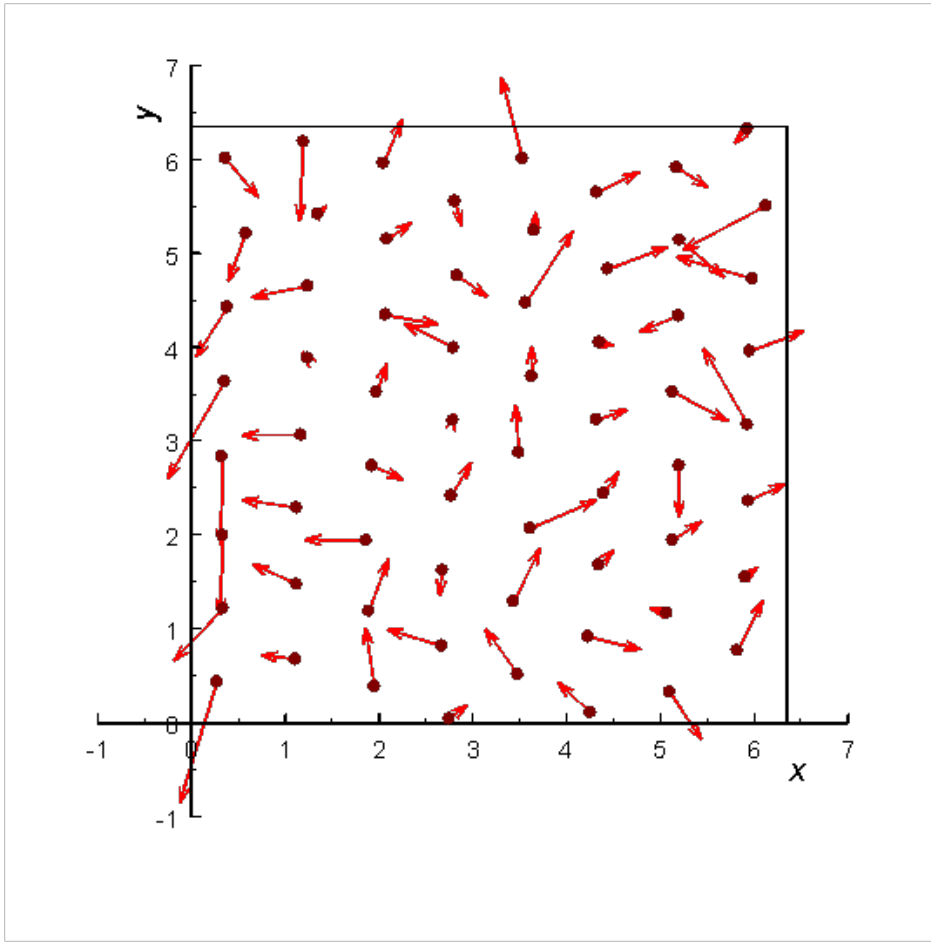
Dinámica Molecular

Busca estudiar el comportamiento de la materia mediante la integración de las ecuaciones de movimiento de las partículas constituyentes; mediante el uso del computador.

Dadas las posiciones, $\vec{r}_i$; y las fuerzas, $\vec{f}_i$; de todas las partículas, se puede encontrar la aceleración, $\vec{a}_i$. Usando algoritmos numéricos se determina la velocidad y los valores siguientes de posición.

Elementos Discretos

Aplicación de los métodos de la dinámica molecular a objetos macroscópicos.



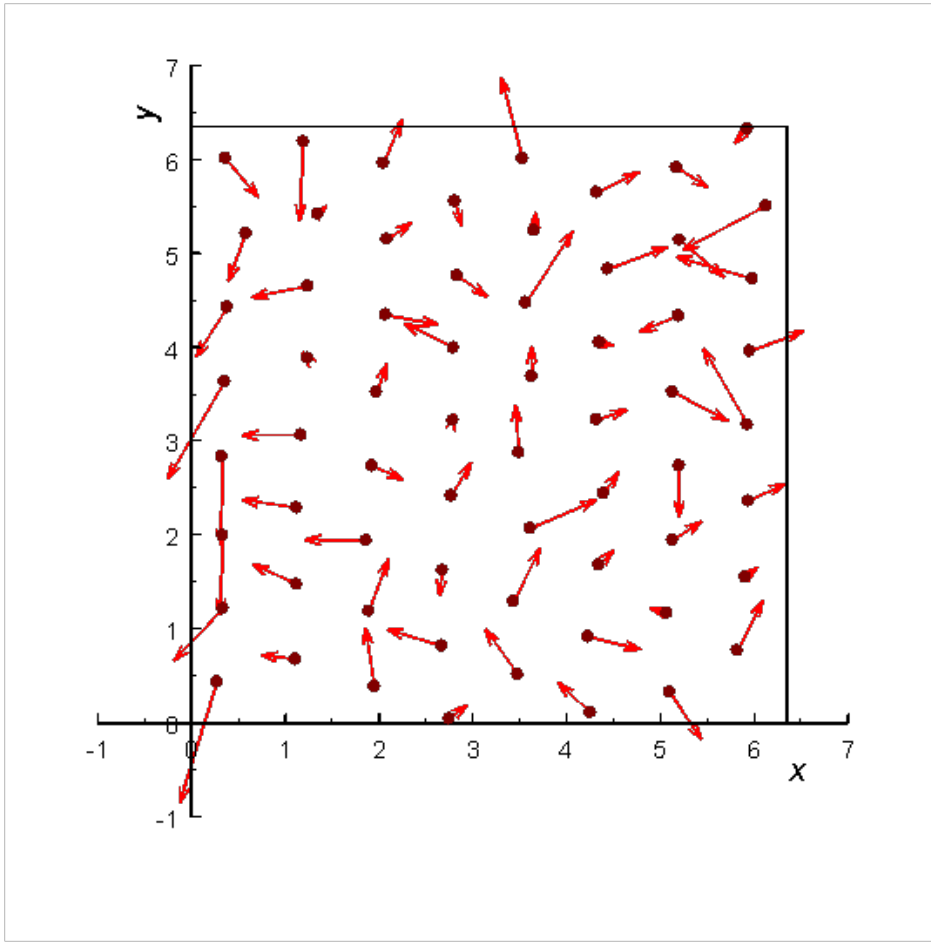
Dinámica molecular

Dinámica Molecular

Busca estudiar el comportamiento de la materia mediante la integración de las ecuaciones de movimiento de las partículas constituyentes; mediante el uso del computador.
Dadas las posiciones, $\vec{r}_i$; y las fuerzas, $\vec{f}_i$; de todas las partículas, se puede encontrar la aceleración, $\vec{a}_i$. Usando algoritmos numéricos se determina la velocidad y los valores siguientes de posición.

Elementos Discretos

Aplicación de los métodos de la dinámica molecular a objetos macroscópicos.



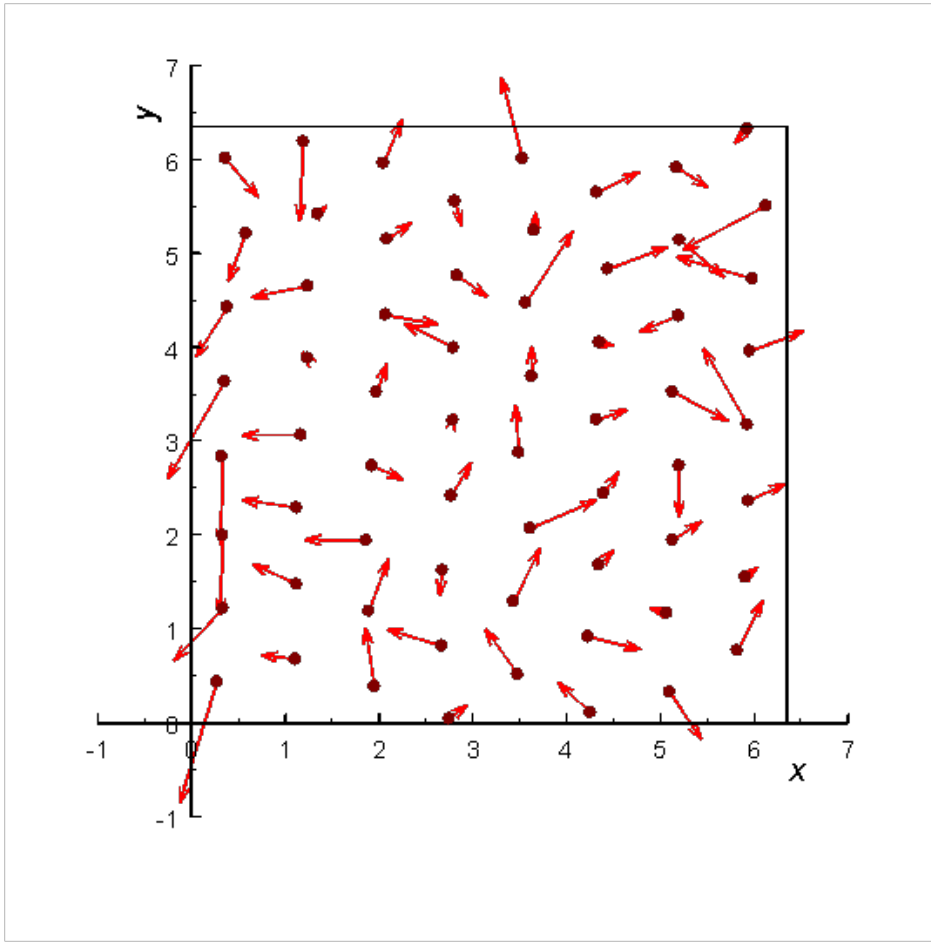
Dinámica molecular

Dinámica Molecular

Busca estudiar el comportamiento de la materia mediante la integración de las ecuaciones de movimiento de las partículas constituyentes; mediante el uso del computador.
Dadas las posiciones, $\vec{r}_i$; y las fuerzas, $\vec{f}_i$; de todas las partículas, se puede encontrar la aceleración, $\vec{a}_i$. Usando algoritmos numéricos se determina la velocidad y los valores siguientes de posición.

Elementos Discretos

Aplicación de los métodos de la dinámica molecular a objetos macroscópicos.

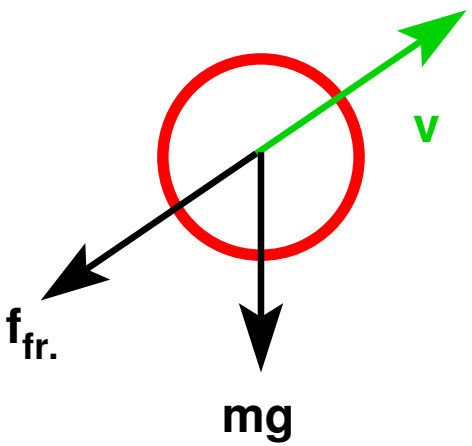


Una sóla pelota que rebota con el suelo

Esfera que rebota

El modelo es un disco que se mueve bajo la acción de la fuerza gravitacional, la fricción viscosa y una repulsión con el suelo proporcional a su área.

En el aire, sólo hay dos fuerzas actuando sobre la pelota: la fricción y el peso.



Si velocidad es $\vec{v}$, entonces la magnitud de la fuerza es:

$$\vec{F}_{fricc.} = -b\vec{v} \quad (1)$$

Al tocar el suelo, por otro lado, existe una fuerza de repulsión. Supongámosla proporcional a la distancia que ha penetrado la esfera con el suelo.

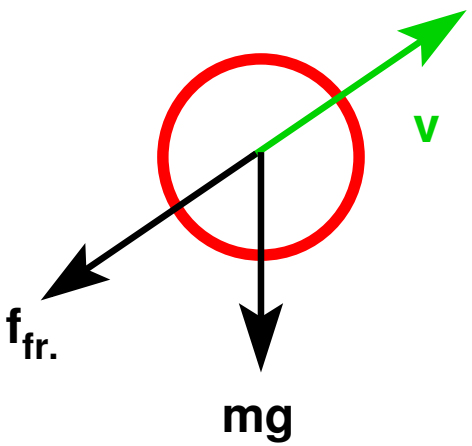
$$F_{rebote} = K * (R - y)\hat{j}$$

Una sóla pelota que rebota con el suelo

Esfera que rebota

El modelo es un disco que se mueve bajo la acción de la fuerza gravitacional, la fricción viscosa y una repulsión con el suelo proporcional a su área.

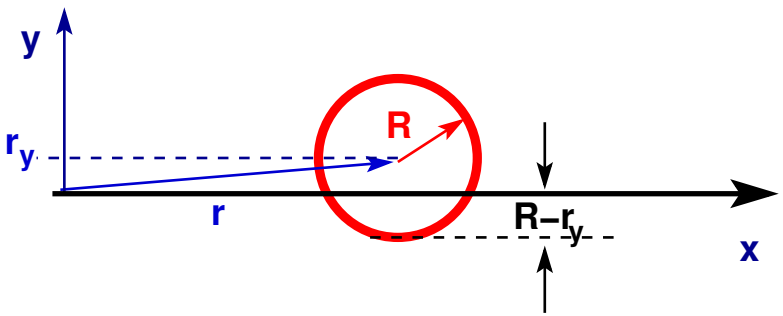
En el aire, sólo hay dos fuerzas actuando sobre la pelota: la fricción y el peso.



Si velocidad es $\vec{v}$, entonces la magnitud de la fuerza es:

$$\vec{F}_{fricc.} = -b\vec{v} \qquad (1)$$

Al tocar el suelo, por otro lado, existe una fuerza de repulsión. Supongámosla proporcional a la distancia que ha penetrado la esfera con el suelo.



$$F_{rebote} = K * (R - y)\hat{j}$$

Integración

Leap Frog

Es un método numérico de integración de segundo orden:

$$x_i = x_{i-1} + v_{i-1/2}\Delta t$$

$$a_i = F(x_i)/m$$

$$v_{i+1/2} = v_{i-1/2} + a_i \Delta t$$

La velocidad y la posición se actualizan en tiempos distintos, la posición depende de la velocidad medio paso de tiempo antes.

Implementación

1. En el tiempo 0, la posición es x_0. Como la velocidad corresponde a medio paso antes, se aproxima usando la fuerza como $\vec{v}_{-0,5} = \vec{v}_0 - 0{,}5\Delta t \vec{F}/m$
2. Repetición: Calcule las fuerzas $\vec{F}$ dadas las posiciones $\vec{x}$.
3. Repetición: Calcule las nuevas velocidades, correspondientes al tiempo $i - 0{,}5$.
4. Repetición: Calcule las nuevas posiciones, correspondientes al tiempo i.
5. Vuelva al paso 2, o termine si $t = i\Delta t = t_{tot}$

Integración

Leap Frog

Es un método numérico de integración de segundo orden:

$$x_i = x_{i-1} + v_{i-1/2}\Delta t$$

$$a_i = F(x_i)/m$$

$$v_{i+1/2} = v_{i-1/2} + a_i\Delta t$$

La velocidad y la posición se actualizan en tiempos distintos, la posición depende de la velocidad medio paso de tiempo antes.

Rana Cristal de roca

Implementación

1. En el tiempo 0, la posición es x_0. Como la velocidad corresponde a medio paso antes, se aproxima usando la fuerza como $\vec{v}_{-0,5} = \vec{v}_0 - 0{,}5\Delta t\vec{F}/m$
2. Repetición: Calcule las fuerzas $\vec{F}$ dadas las posiciones $\vec{x}$.
3. Repetición: Calcule las nuevas velocidades, correspondientes al tiempo $i - 0{,}5$.
4. Repetición: Calcule las nuevas posiciones, correspondientes al tiempo i.
5. Vuelva al paso 2, o termine si $t = i\Delta t = t_{tot}$

Integración

Leap Frog

Es un método numérico de integración de segundo orden:

$$x_i = x_{i-1} + v_{i-1/2}\Delta t$$

$$a_i = F(x_i)/m$$

$$v_{i+1/2} = v_{i-1/2} + a_i\Delta t$$

La velocidad y la posición se actualizan en tiempos distintos, la posición depende de la velocidad medio paso de tiempo antes.

Rana Cristal de roca

Implementación

1. En el tiempo 0, la posición es x_0. Como la velocidad corresponde a medio paso antes, se aproxima usando la fuerza como $\vec{v}_{-0,5} = \vec{v}_0 - 0{,}5\Delta t\vec{F}/m$
2. Repetición: Calcule las fuerzas $\vec{F}$ dadas las posiciones $\vec{x}$.
3. Repetición: Calcule las nuevas velocidades, correspondientes al tiempo $i - 0{,}5$.
4. Repetición: Calcule las nuevas posiciones, correspondientes al tiempo i.
5. Vuelva al paso 2, o termine si $t = i\Delta t = t_{tot}$

Integración

Leap Frog

Es un método numérico de integración de segundo orden:

$$x_i = x_{i-1} + v_{i-1/2}\Delta t$$

$$a_i = F(x_i)/m$$

$$v_{i+1/2} = v_{i-1/2} + a_i\Delta t$$

La velocidad y la posición se actualizan en tiempos distintos, la posición depende de la velocidad medio paso de tiempo antes.

Rana Cristal de roca

Implementación

1. En el tiempo 0, la posición es x_0. Como la velocidad corresponde a medio paso antes, se aproxima usando la fuerza como $\vec{v}_{-0,5} = \vec{v}_0 - 0{,}5\Delta t\vec{F}/m$
2. Repetición: Calcule las fuerzas $\vec{F}$ dadas las posiciones $\vec{x}$.
3. Repetición: Calcule las nuevas velocidades, correspondientes al tiempo $i - 0{,}5$.
4. Repetición: Calcule las nuevas posiciones, correspondientes al tiempo i.
5. Vuelva al paso 2, o termine si $t = i\Delta t = t_{tot}$

Integración

Leap Frog

Es un método numérico de integración de segundo orden:

$$x_i = x_{i-1} + v_{i-1/2}\Delta t$$

$$a_i = F(x_i)/m$$

$$v_{i+1/2} = v_{i-1/2} + a_i\Delta t$$

La velocidad y la posición se actualizan en tiempos distintos, la posición depende de la velocidad medio paso de tiempo antes.

Rana Cristal de roca

Implementación

1. En el tiempo 0, la posición es x_0. Como la velocidad corresponde a medio paso antes, se aproxima usando la fuerza como $\vec{v}_{-0,5} = \vec{v}_0 - 0{,}5\Delta t\vec{F}/m$
2. Repetición: Calcule las fuerzas $\vec{F}$ dadas las posiciones $\vec{x}$.
3. Repetición: Calcule las nuevas velocidades, correspondientes al tiempo $i - 0{,}5$.
4. Repetición: Calcule las nuevas posiciones, correspondientes al tiempo i.
5. Vuelva al paso 2, o termine si $t = i\Delta t = t_{tot}$

Integración

Leap Frog

Es un método numérico de integración de segundo orden:

$$x_i = x_{i-1} + v_{i-1/2}\Delta t$$

$$a_i = F(x_i)/m$$

$$v_{i+1/2} = v_{i-1/2} + a_i\Delta t$$

La velocidad y la posición se actualizan en tiempos distintos, la posición depende de la velocidad medio paso de tiempo antes.

Rana Cristal de roca

Implementación

1. En el tiempo 0, la posición es x_0. Como la velocidad corresponde a medio paso antes, se aproxima usando la fuerza como $\vec{v}_{-0,5} = \vec{v}_0 - 0{,}5\Delta t\vec{F}/m$
2. Repetición: Calcule las fuerzas $\vec{F}$ dadas las posiciones $\vec{x}$.
3. Repetición: Calcule las nuevas velocidades, correspondientes al tiempo $i - 0{,}5$.
4. Repetición: Calcule las nuevas posiciones, correspondientes al tiempo i.
5. Vuelva al paso 2, o termine si $t = i\Delta t = t_{tot}$

Integración

Leap Frog

Es un método numérico de integración de segundo orden:

$$x_i = x_{i-1} + v_{i-1/2}\Delta t$$

$$a_i = F(x_i)/m$$

$$v_{i+1/2} = v_{i-1/2} + a_i\Delta t$$

La velocidad y la posición se actualizan en tiempos distintos, la posición depende de la velocidad medio paso de tiempo antes.

Rana Cristal de roca

Implementación

1. En el tiempo 0, la posición es x_0. Como la velocidad corresponde a medio paso antes, se aproxima usando la fuerza como $\vec{v}_{-0,5} = \vec{v}_0 - 0{,}5\Delta t\vec{F}/m$
2. Repetición: Calcule las fuerzas $\vec{F}$ dadas las posiciones $\vec{x}$.
3. Repetición: Calcule las nuevas velocidades, correspondientes al tiempo $i - 0{,}5$.
4. Repetición: Calcule las nuevas posiciones, correspondientes al tiempo i.
5. Vuelva al paso 2, o termine si $t = i\Delta t = t_{tot}$

Animaciones con gnuplot

Gnuplot Interactivo

En el modo interactivo (la forma usual que hemos usado de llamar a gnuplot escribiendo "gnuplot", lo cual nos da un prompt `gnuplot>`), se pueden introducir las instrucciones para generar gráficas:

```
set xlabel tiempo
set ylabel posicion
set title ``un titulo muy
informativo''
set term png
set output archivo.png
plot data.txt
```

Más información

http://lowrank.net/gnuplot/index-e.html

c++ + gnuplot + pipes

Una forma de hacer una animación con `gnuplot` consiste en construir un programa de computador que le alimente las instrucciones paso a paso, usando pipes. Las instrucciones:

`unset key` : Remueve la leyenda

`set xrange[-10:250]` : Límites eje x

`set yrange[-10:50]` : Límites eje y

`set size ratio -1` : Distancias en y iguales a las de x

`set parametric` : Coord. paramétricas

`set trange [0:7]` : Rango de t en las paramétricas

Animaciones con gnuplot

Gnuplot Interactivo

En el modo interactivo (la forma usual que hemos usado de llamar a gnuplot escribiendo "gnuplot", lo cual nos da un prompt `gnuplot>`), se pueden introducir las instrucciones para generar gráficas:

```
set xlabel tiempo
set ylabel posicion
set title ``un titulo muy informativo''
set term png
set output archivo.png
plot data.txt
```

Más información

http://lowrank.net/gnuplot/index-e.html

c++ + gnuplot + pipes

Una forma de hacer una animación con `gnuplot` consiste en construir un programa de computador que le alimente las instrucciones paso a paso, usando pipes. Las instrucciones:

`unset key` : Remueve la leyenda

`set xrange[-10:250]` : Límites eje x

`set yrange[-10:50]` : Límites eje y

`set size ratio -1` : Distancias en y iguales a las de x

`set parametric` : Coord. paramétricas

`set trange [0:7]` : Rango de t en las paramétricas

Clase Grano

Clase Grano

```
class Grano{
private:
  double x,y,Vx,Vy,Fx,Fy,R,m;
  double EH;
public:
  void Inicie(double x0,double y0,double Vx0,double Vy0,
double R0,double m0);
  void CalculeFuerza(void);
  void Arranque(double dt);
  void Muevase(double dt);
  double Getx(void){return x;};
  double Gety(void){return y;};
  double GetE(void) {return 0.5*m*(Vx*Vx+Vy*Vy) + m*g*y
+ EH;}
  void Dibujese(void);
};
void Grano::Inicie(double x0,double y0,double Vx0,double
 Vy0,double R0,double m0){
  x=x0; y=y0; Vx=Vx0; Vy=Vy0; R=R0; m=m0;
}
```

Las funciones gnuplot:

Clase Grano

Clase Grano

```
class Grano{
private:
  double x,y,Vx,Vy,Fx,Fy,R,m;
  double EH;
public:
  void Inicie(double x0,double y0,double Vx0,double Vy0,
double R0,double m0);
  void CalculeFuerza(void);
  void Arranque(double dt);
  void Muevase(double dt);
  double Getx(void){return x;};
  double Gety(void){return y;};
  double GetE(void) {return 0.5*m*(Vx*Vx+Vy*Vy) + m*g*y
+ EH;}
  void Dibujese(void);
};
void Grano::Inicie(double x0,double y0,double Vx0,double
 Vy0,double R0,double m0){
  x=x0; y=y0; Vx=Vx0; Vy=Vy0; R=R0; m=m0;
}
```

Las funciones gnuplot:

```
void InicieAnimacion(void){
  cout<<"unset key"<<endl;
  cout<<"set xrange[-10:250]"<<endl;
  cout<<"set yrange[-10:50]"<<endl;
  cout<<"set size ratio -1"<<endl;
  cout<<"set parametric"<<endl;
  cout<<"set trange [0:7]"<<endl;
  cout<<"set isosamples 12"<<endl;
}
```

Clase Grano

Clase Grano

```
class Grano{
private:
  double x,y,Vx,Vy,Fx,Fy,R,m;
  double EH;
public:
  void Inicie(double x0,double y0,double Vx0,double Vy0,
double R0,double m0);
  void CalculeFuerza(void);
  void Arranque(double dt);
  void Muevase(double dt);
  double Getx(void){return x;};
  double Gety(void){return y;};
  double GetE(void) {return 0.5*m*(Vx*Vx+Vy*Vy) + m*g*y
+ EH;}
  void Dibujese(void);
};
void Grano::Inicie(double x0,double y0,double Vx0,double
 Vy0,double R0,double m0){
  x=x0; y=y0; Vx=Vx0; Vy=Vy0; R=R0; m=m0;
}
```

Las funciones gnuplot:

```
void InicieAnimacion(void){
  cout<<"unset key"<<endl;
  cout<<"set xrange[-10:250]"<<endl;
  cout<<"set yrange[-10:50]"<<endl;
  cout<<"set size ratio -1"<<endl;
  cout<<"set parametric"<<endl;
  cout<<"set trange [0:7]"<<endl;
  cout<<"set isosamples 12"<<endl;
}
```

```
void IniciePantalla(void){
    cout<<"plot 0,0 ";
}
```

Clase Grano

Clase Grano

```
class Grano{
private:
  double x,y,Vx,Vy,Fx,Fy,R,m;
  double EH;
public:
  void Inicie(double x0,double y0,double Vx0,double Vy0,
double R0,double m0);
  void CalculeFuerza(void);
  void Arranque(double dt);
  void Muevase(double dt);
  double Getx(void){return x;};
  double Gety(void){return y;};
  double GetE(void) {return 0.5*m*(Vx*Vx+Vy*Vy) + m*g*y
+ EH;}
  void Dibujese(void);
};
void Grano::Inicie(double x0,double y0,double Vx0,double
 Vy0,double R0,double m0){
  x=x0; y=y0; Vx=Vx0; Vy=Vy0; R=R0; m=m0;
}
```

Las funciones gnuplot:

```
void InicieAnimacion(void){
  cout<<"unset key"<<endl;
  cout<<"set xrange[-10:250]"<<endl;
  cout<<"set yrange[-10:50]"<<endl;
  cout<<"set size ratio -1"<<endl;
  cout<<"set parametric"<<endl;
  cout<<"set trange [0:7]"<<endl;
  cout<<"set isosamples 12"<<endl;
}
```

```
void IniciePantalla(void){
    cout<<"plot 0,0 ";
}
```

```
void TerminePantalla(void){
  cout << ", 250*t/7.,0 " << endl;
  cout<<endl;
}
```

vacio

Integración Leap-Frog

```
void Grano::Arranque(double dt)
{
  Vx = Vx - 0.5*dt*(Fx/m);
  Vy = Vy - 0.5*dt*(Fy/m);
}
```

Tarea:

vacio

Integración Leap-Frog

```
void Grano::Arranque(double dt)
{
  Vx = Vx - 0.5*dt*(Fx/m);
  Vy = Vy - 0.5*dt*(Fy/m);
}
```

```
void Grano::CalculeFuerza(void){
  EH=0;
  Fx=0; Fy=-m*g;
  // calcular la fuerza con el suelo
  double S = R - y;
  if (S > 0.0) {
    EH += 0.5*K*S*S;
    Fy += K*S;  // resorte
    Fy -= b*Vy; // viscosa
  }
}
```

Tarea:

vacio

Integración Leap-Frog

```
void Grano::Arranque(double dt)
{
  Vx = Vx - 0.5*dt*(Fx/m);
  Vy = Vy - 0.5*dt*(Fy/m);
}
```

```
void Grano::CalculeFuerza(void){
  EH=0;
  Fx=0; Fy=-m*g;
  // calcular la fuerza con el suelo
  double S = R - y;
  if (S > 0.0) {
    EH += 0.5*K*S*S;
    Fy += K*S;  // resorte
    Fy -= b*Vy; // viscosa
  }
}
```

```
void Grano::Muevase(double dt){
  Vx = Vx + dt*(Fx/m);
  x = x + dt*Vx;
  Vy = Vy + dt*(Fy/m);
  y = y + dt*Vy;
}
```

Tarea: