Guion de prácticas

Proyecto Final - Parte A

Metodología de la Programación

Curso 2017/2018

Introducción

Este guion contiene las instrucciones para el desarrollo del proyecto final de la asignatura. Se requiere disponer de la implementación de la clase `Particula` planteada en el guión anterior. Se construirá una clase completa `ConjuntoParticulas` para representar un conjunto de objetos de la clase `Particula` (implementada en el guion anterior). Por el momento, la visualización se seguirá realizando directamente en el `main`.

Clase Particula

Extienda la clase Particula con los siguientes métodos:

1. `Mover(int alto, int ancho)`: cambia la posición de la partícula. El método tendrá en cuenta los rebotes contra los bordes en un espacio que va de las coordenadas `(0,0)` hasta `(alto, ancho)`. Para evitar complicaciones, supondremos que el punto (0,0) representa el vértice superior izquierdo, mientras que `(alto, ancho)`, es el inferior derecho (es decir, la posición de la partícula se corresponde con la posición de la ventana de MiniWin). En la práctica anterior, esta tarea se realizaba con una función.

2. `Rebota(Particula & otra)`: implementa el choque elástico entre dos partículas. Asumiendo que ambas tienen la misma "masa", el choque elástico implica intercambiar las respectivas velocidades (investigue el por qué).

Clase ConjuntoParticulas

Se definirá una clase que permita representar una conjunto de objetos de tipo `Particula`. Se propone la siguiente definición:

```
class ConjuntoParticulas{
private:
  Particula *set;    // un array de particulas
  int capacidad;       // capacidad del array
  int utiles;          // posiciones ocupadas
```

Respecto a los métodos, serán necesarios:

1. Constructor con parámetro: recibe un entero representando la capacidad C del conjunto e inicializa el array con C partículas al azar.

2. Constructor de copia.

3. Destructor: libera la memoria reservada.

4. Métodos Get para capacidad y elementos útiles del conjunto.

5. `AgregaParticula`: agrega una partícula al conjunto. Si no hay espacio suficiente, se redimensiona el array extendiendo su capacidad en `TAM_BLOQUE` unidades. Defina la constante `TAM_BLOQUE = 5` en el fichero `ConjuntoParticulas.h`.

6. `BorraParticula`: haciendo desplazamientos a izquierda, elimina la partícula de una posición dada. Si luego del borrado, se verifica que `(capacidad - utiles) >TAM_BLOQUE`, entonces debe redimensionar el conjunto para que la nueva capacidad coincida con el número de partículas.

7. `ObtieneParticula`: devuelve la partícula de una posición dada.

8. `ReemplazaParticula`: reemplaza la partícula de una posición dada con otra.

9. `Mover`: recibe como parámetros los valores de ancho y alto en los que las partículas se pueden mover. Luego, mueve cada partícula usando el método correspondiente.

10. `GestionarColisiones`: Evalúa todos los pares de partículas. Aquellas que colisionan, "rebotan" con el método correspondiente.

11. Sobrecarga de operadores. Implemente:

    - Operador de asignación.
    - Operador $<<$ con una función externa (utilice como referencia los ejemplos mostrados en las transparencias de teoría.)
    - Operador '+' para concatenar dos objetos de la clase `ConjuntoParticulas`.

Para probar la clase utilice el código mostrado en la Fig. 1 (el código fuente está disponible en PRADO). Más adelante se proveerá un nuevo código para probar los operadores sobrecargados.

Observaciones

El programa debe estar correctamente modularizado y organizado en las carpetas `src, include`. Cada clase debe tener un fichero `.h` y el correspondiente `.cpp`.

El programa se evaluará con la herramienta `valgrind` (documento disponible en PRADO).

Sea cuidadoso con la implementación. Tenga en cuenta la indentación, piense cuidadosamente el tipo de los parámetros que reciben y devuelven los métodos, no repita código y utilice métodos privados para evitarlo (por ejemplo para `reservarMemoria, liberarMemoria, redimensiona`,etc.).

RECUERDE: Estas instrucciones deben complementarse con las indicaciones dadas durante las sesiones de práctica.

```
void pintaNube(const ConjuntoParticulas & miConjunto) {
  int N = miConjunto.GetNroParticulas();
  Particula p;
  for (int i = 0; i < N; i++) {
    p = miConjunto.ObtieneParticula(i);
    color(p.getColor());
    circulo_lleno(p.getX(), p.getY(), RADIO);
  }
}

int main() {
srand(time(0));

int ancho = 800;
int alto = 800;
vredimensiona(ancho, alto);

int contador = 0;
int minParticulas = 2;
int maxParticulas = 50;

ConjuntoParticulas nube(minParticulas);
bool AGREGA = true;
int nroParticulas;

while (tecla() != ESCAPE) {
  nube.Mover(ancho, alto);
  nube.GestionarColisiones();
  borra();
  pintaNube(nube);
  contador++;

  nroParticulas = nube.GetNroParticulas();
  color(BLANCO);
  texto(10,10, "Nro Particulas: " + to_string(nroParticulas));

  // agregar o borrar particulas cada 10 iteraciones
  if (contador %10 == 0){
    if (AGREGA){
      Particula p;
      nube.AgregaParticula(p);
    }
  else
    // borro una particula al azar
    nube.BorraParticula(rand()%nroParticulas);
  }

  if (nroParticulas <= minParticulas)
    AGREGA = true;
  else if (nroParticulas >= maxParticulas)
    AGREGA = false;

  refresca();
  espera(25);
  }

vcierra();
return 0;
}
```

Figura 1: Modelo de programa para probar la clase `ConjuntoParticulas`.