Nombre: _____
Código: _____

1. Introducción

Este parcial consta de 4 puntos cada uno con el mismo valor porcentual para el cálculo de la nota.

1.1. Internet

Las sala permanecerán bloqueadas a internet, excepto a los siguientes recursos:

Lugar	**Enlace**
Riouxsvn	`https://riouxsvn.com`
C++ Reference	`http://en.cppreference.com/w/`
C++ Reference	`http://www.cplusplus.com/`
Pagina curso	`http://www1.eafit.edu.co/fcardona/`

1.2. Instalacion

Abra una sesión de cygwin y ejecute los siguiente:

```
$ mkdir st0244
$ cd st0244
$ svn co https://svn.riouxsvn.com/244s<username>
$ cd 244s<username>/
```

Si no esta creado cree el directorio `parciales`

```
$ svn mkdir parciales
$ svn mkdir parciales/parcial02b
$ cd parciales/parcial02b
$ wget http://www1.eafit.edu.co/fcardona/cursos/st0244/parcial02b.zip
```

Ahora encontrara todas las carpetas del proyecto:

Una copia del parcial esta en `http://www1.eafit.edu.co/fcardona/cursos/st0244/Parcial02b.pdf`.

1.3. Configuración `emacs` y `mintty`

Una configuración básica de emacs se encuentra en `http://www1.eafit.edu.co/fcardona/cursos/st0244/emacs` y una configuración básica de curl se encuentra en `http://www1.eafit.edu.co/fcardona/cursos/st0244/minttyrc`. Abra una terminal nueva y ejecute:

```
$ curl http://www1.eafit.edu.co/fcardona/cursos/st0244/emacs > .emacs
$ curl http://www1.eafit.edu.co/fcardona/cursos/st0244/minttyrc > .minttyrc
```

2. Preguntas

1. (25 points) **Intercalate**

   (directorio: `intercalate` fichero: `intercalate.cpp` Implementar la función `intercalate`

```
int* intercalate(const int arr1[],
      const int arr2[],
      const int nbr,
      const int fact);
```

   que toma dos arreglos (`arr1` y `arr2`) del mismo tamaño (`nbr`) y produce un nuevo arreglo de mismo tamaño que tiene una versión de los arreglos intercalados dependiendo de un factor (`fact`) que es un divisor del tamaño de arreglo.

   Por ejemplo con dos arreglos de 4 elementos: $[1, 2, 3, 4]$ y $[5, 6, 7, 8]$ y un factor de intercalación 2 se produce el siguiente arreglo: $[6, 1, 8, 3]$. Con los mismos arreglos y una factor de intercalación de 4 se produce el siguiente arreglo: $[7, 8, 1, 2]$ y finalmente los mismos arreglos y un factor de intercalación de 4 se produce: $[5, 6, 7, 8]$.

2. (25 points) **Parity**

   (directorio: `parity` fichero: `parityTest.cpp`) Implemente la función `parityTest`:

```
enum Parity {odd, even, none};

Parity parityTest(const char word[], int& nEven, int &nOdd);
```

   La función `parityTest` recibe una cadena de caracteres constantes (`word`) y dos referencias a enteros(`nEven` y `nOdd`)donde se retorna el número de caracteres con paridad par e impar respectivamente. Esta función retorna el tipo de paridad: par (`even`) si hay un número mayor de elementos con paridad par, impar (`odd`) si hay un número mayor de elemento con paridad inmpar y `none` si los números de elementos son iguales. Cada caracter es examinado con la función `isEvenParity` para ver si el caracter es de paridad par o `isOddParity` para ver si el caracter es de paridad impar. Las funciones `isEvenParity` o `isOddParity` no requiere ser implementadas y se encuentran declaradas

3. (25 points) ***Game simulator***
(directorio: `player`, fichero: `player.cpp`) Se implementará la siguiente clase `Player` que modela el comportamiento de un jugador en un supuesto simulador de juego que se está implementado:

```
class Player {
 public:
  Player(unsigned short nLife,
         unsigned short nBullets,
         unsigned short armorLevel,
         unsigned short shotFactor);
  void setLifes(unsigned short nLifes);
  void setArmorLevel(unsigned short level);
  void chargeCartridge(unsigned short nBullets);
  void receiveShot();
  void fireMachineGun();
  bool isAlive() const;
  unsigned short getLifes() const;
  unsigned short getArmorLevel() const;
  unsigned short getNbrBullets() const;
 private:
  unsigned short nLifes;
  unsigned short nBullets;
  unsigned short armorLevel;
  unsigned short shotFactor;
};
```

La clase `Player` tiene los siguientes atributos: `nLifes` el número de vidas que tiene el jugador, `nBullets` el número de balas que guarda en un su magazin, `armorLevel` el nivel de armadura, `shotFactor` un factor que indica en cuanto disminuye la armadura por cada disparo que recibe.

Cuando un jugador es creado recibe un cantidad determinada de vida, de balas, nivel de armadura y factor del disparo. A trancurrir el juego el jugador obtendrá vidas (`setLifes`), aumentará las balas que lleva en el magazin (`chargeCartridge`), también adquirirá más armadura (`setArmorLevel`).

Pero el jugador también dispara su arma (`fireMachineGun`) disparando una bala a la vez[1]. Pero también recibirá disparos que afectarán su armadura (si tiene) y sus vidas (si tiene), cada disparo disminuye la armadura en el factor de disparo (`shotFactor`), en el momento que no tenga más armadura, perderá un vida y cada disparo subsecuente perderá vidas, hasta morir. Es evidente, pero un jugador no puede recibir ni vidas, ni balas, ni armadura, después de muerto.

[1] ¿Teniendo una *machine gun*?

4. (25 points) ***fair dicharge bateries***

   (directorio: `bateries` fichero: `bateries.cpp`) Se están diseñando un sistema de baterias que permiten descargar propocionalmente las baterias de modo que las baterias se descargan homogénamente, así se espera que las baterías alcancen un desgate homogeneo en su tiempo de vida.

   Para ello se ha diseñado un sistema que se presenta en la figura 1:

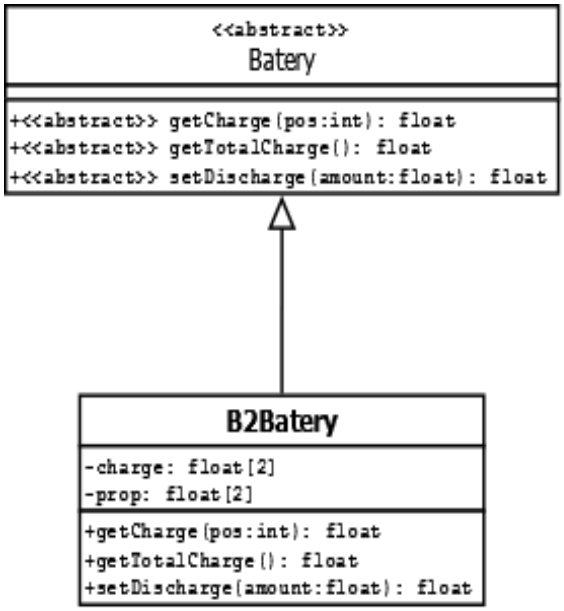

Figura 1: Jeraquía de directorios `parcial02`

Donde la clase `Batery` representa todas las posibles clase de baterias que se pueden tener y `B2Batery` es una versión concreta de la clase de bateria que implementa un arreglo interno de 2 celdas (`charge`) para guardar la energía. En la bateria `B2Bateryla` suma de las dos celdas representa la carga total de una bateria (`getTotalCharge`). `B2Batery` se descarga proporcionalmente (`prop`: es un arreglo que guarda la proporción $charge_i/totalCharge$ de cada celda) dado el peso (proporción) de cada celda con respecto al monto total. El calculo de la carga de cada celda ($carga_i$) se calcula así:

$$carga_i \leftarrow amount \times prop_i$$

Se debe tener en cuenta que una bateria descargada es decir $carga_i = 0$ no se sigue descargando más.

La proporción $prop_i$ se calcula:

$$prop_i \leftarrow \frac{carga_i}{total_{bateria}}$$