1. **CapitalizeFirst**: Realice una función que reciba un std::string y "capitalice" cada inicio de palabra del mismo. Por ejemplo si el texto original es "este es el primer parcial de introducción al cómputo" el resultado deberá ser: "Este Es El Primer Parcial De Introducción Al Cómputo". El prototipo de la función deberá ser:

```
void CapitalizeFirst(std::string &phrase);
```

2. **Validación de CBU**: La Clave Bancaria Uniforme (CBU) es un código utilizado en Argentina por los bancos para la identificación de las cuentas de sus clientes. Mientras que cada banco identifica las cuentas con números internos, la CBU identifica una cuenta de manera única en todo el sistema financiero argentino. Está compuesta por 22 dígitos, separados en dos bloques (de 8 y 14 dígitos respectivamente). Por ejemplo:

   28505909 40090418135201

   En el primer bloque, los primeros 3 dígitos corresponden al Banco. Luego le sigue el primer "dígito verificador". Los siguientes 3 dígitos identifican la sucursal y por último el segundo dígito verificador.

   El segundo bloque se compone de 13 dígitos para el número de cuenta y el tercer y último dígito verificador.

   En el ejemplo dado sería:

   Banco: **285** (B_0=2, B_1=8, B_2=5) DV_0 = 0

   Sucursal: **590** (S_0=5, S_1=9, S_2=0) DV_1 = 9

   Cuenta: **4009041813520** (C_0=4, C_1=0, C_2=0, …) DV_2 = 1

   El procedimiento para validar una clave CBU es el siguiente:

   - Validación del primer bloque: Se deben sumar los dígitos correspondientes al Banco, DV_0, y Sucursal, con los siguientes pesos: (7, 1, 3, 9, 7, 1, 3). En el ejemplo sería B_0*7 + B_1*1 + B_2*3 + DV_0*9 + S_0*7 + S_1*1 + S_2*3=81. El último dígito de esta suma se resta de 10 (En el ejemplo: DIF_1=10-1=9) y este valor debe coincidir con el DV_1 (En el ej. DV_1=9=DIF_1).
   - Validación del segundo bloque: Sumar los dígitos del número de cuenta, con los siguientes pesos: (3, 9, 7, 1, 3, 9, 7, 1, 3, 9, 7, 1, 3). En el ejemplo sería (C_0*3 + C_1*9 + C_2*7 + C_3*1 + C_4*3 + C_5*9 + C_6*7 + C_7*1 + C_8*3 + C_9*9 + C_{10}*7 + C_{11}*1 + C_{12}*3=139). El último dígito de esta suma se resta de 10 (En el ejemplo: DIF_2=10-9=1) y este valor debe coincidir con el DV_2 (En el ej. DV_2=1=DIF_2).

   Implemente una función que reciba una CBU en forma de string nativo, y lo verifique. Debe retornar verdadero o falso. Respete el siguiente prototipo:

```
int cbu_check(const char cbu[]);
```

   Implemente un programa que solicite al usuario un número de CBU, lo verifique e imprima por pantalla si el número es válido o no. Implemente la función de forma que resulte sencillo cambiar los pesos que se le da a cada dígito.

3. **Simulación de un cardumen:** Es posible simular el comportamiento de cardumen con tan solo 3 reglas muy simples que afectan la velocidad de cada pez:

   a. Tender a acercarse al centro ($\boldsymbol{rc} = \frac{\sum_j^{N_p} \boldsymbol{r_j}}{N_p}$) del cardumen $\delta \boldsymbol{v1}_i = \frac{\boldsymbol{rc} - \boldsymbol{r_i}}{8}$

b. Evitar chocar con los vecinos más cercanos $\delta v2_i = \sum_j^{|r_j - r_i|<3} \frac{r_i - r_j}{|r_j - r_i|}$ (solo se tienen en cuenta los peces que están a una distancia menor a una distancia dada (**`maxDist`**))

c. Tender a igualar la velocidad media ($\boldsymbol{vc} = \frac{\sum_j^{N_p} v_j}{N_p}$) del cardumen $\delta v3_i = \frac{vc - v_i}{8}$

El cambio en la velocidad del pez i-ésimo será la suma de los cambios debido a estas 3 reglas $\delta v_i = \delta v1_i + \delta v2_i + \delta v3_i$ y la velocidad de cada pez no puede superar una velocidad máxima dada (**`maxVel`**). Con esta velocidad se calcula la evolución temporal de la posición $r_i(t + \delta t) = r_i(t) + v_i(t) \times \delta t$

Por simplicidad vamos a limitarnos a 2D, los pasos temporales son de a 1 unidad y representaremos a nuestras entidades con las siguientes estructuras y definiciones:

```
struct V2_t {       // posición o velocidad en 2D
    double x, y;
    //...
};
struct Pez_t {      // representa un pez: posición y velocidad en 2D
    V2_t pos;
    V2_t vel;
    //...
};
```

En el programa, un cardumen será representado por una instancia del tipo Cardumen:

```
struct Cardumen {

    double maxVel;
    double maxDist;

    static const int MAX_PECES = 16;
    std::array<Pez_t, MAX_PECES> peces;

    // Inicializa todos los peces con  posiciones al azar dentro de un área
    // cuadrada de n x n y con velocidades al azar con 0 < |v| < maxVel.
    void initialize(int n, double maxVel_, double maxDist_);

    // Calcula y retorna el centro de masa de los peces(para su uso en R1)
    V2_t calculateCM();

    // Calcula y retorna la veloidad media de todos peces(para su uso en R3)
    V2_t calculateVM();

    // aplica la regla 1 al pez i retornando su deltaV por R1
    V2_t applyR1(int i, V2_t cm);

    // aplica la regla 2 al pez i retornando su deltaV por R2
    V2_t applyR2(int i);

    // aplica la regla 3 al pez i retornando su deltaV por R3
    V2_t applyR3(int I, V2_t vm);

    // hace evolucionar el cardumen 1 paso de tiempo. Calcula CM, VM y las
    // nuevas velocidades de todos los peces a partir de las aplicaciones
    // de las reglas 1, 2 y 3 aplicando además el limitador de velocidad.
    // Actualiza las velocidades y posiciones de todos los peces del cardumen.
    void doStep();

    // Imprime el estado actual de cardumen. (Posiciones y velocidades
    void print();

};
```

Se solicita implementar los métodos anteriores y probarlo con:

```
int main()
{
    const int NUM_PASOS = 100;
    Cardumen c;

    c.initialize(40, 4, 3);
    for (int i = 0; i < NUM_PASOS; ++i) {
       c.doStep();
       c.print();
    }
    return 0;
}
```

Nota 1: Tenga en cuenta que en las 3 reglas descriptas, las posiciones (r_j) y velocidades (v_j) están representados por un **`V2_t`**, no por escalares.