Deadline

```
// Pre: 0 <= time_1 <= 60 and 0 <= time_2 <= 60 are two consecutive values
//              in the timestamp sequence
// Post: returns the elapsed time between time_1 and time_2
int time_lapse(int time_1, int time_2) {
    if (time_2 > time_1) return time_2 - time_1;
    else return 60 + time_2 - time_1;
}

int main() {
    int k;
    cin >> k;
    int time;
    cin >> time;
    int position = 1;
    int new_time;
    cin >> new_time;
    while (new_time != -1 and k >= 0) {
        k -= time_lapse(time, new_time);
        ++position;
        time = new_time;
        cin >> new_time;
    }
    if (k < 0) cout << position << endl;
    else cout << 0 << endl;
}
```

Última posición del primer elemento repetido

```
// Pre: --
// Post: retorna true si v tiene algún elemento repetido; x es el primer elemento repetido
//         en la secuencia y j es la última posición en la que aparece en la secuencia;
//       retorna false, en caso contrario (y no importa el valor de x y j)
bool hay_repetidos(const vector <int>& v, int& x, int& j) {
    int i = 0;
    bool encontrado = false;
    while (not encontrado and i < v.size()) {
        x = v[i];
        j = v.size() - 1;
        while (x != v[j]) --j;
        if (i != j) encontrado = true;
        else ++i;
    }
    return encontrado;
}

// Pre: en la entrada hay n valores enteros
// Post: retorna un vector que almacena los n enteros de la entrada manteniendo el orden
vector <int> leer_secuencia(int n) {
    vector <int> v(n);
    for (int i = 0; i < n; ++i) cin >> v[i];
    return v;
}

int main() {
    int n;
    while (cin >> n) {
        vector <int> v = leer_secuencia(n);
        int x, i;
        if (hay_repetidos(v, x, i)) cout << x << " " << i << endl;
        else cout << "NO" << endl;
    }
}
```

Factor alfa

```
struct Asignatura {
    int id;
    double creditos;
    double nota;
};

struct Alumno {
    int dni;
    vector <Asignatura> asigs;
};

typedef vector <Alumno> Alumnos;

// Pre: --
// Post: retorna el número de alumnos en alus que superan el factor alfa
int supera_alfa(const Alumnos& alus) {
    int a = 0;
    for (int i = 0; i < alus.size(); ++i) {
        double creditos = 0;
        double suma = 0;
        for (int j = 0; j < alus[i].asigs.size(); ++j) {
            suma += alus[i].asigs[j].nota*alus[i].asigs[j].creditos;
            creditos += alus[i].asigs[j].creditos;
        }
        if (creditos != 0 and suma/creditos >= 5) ++a;
    }
    return a;
}

// Pre: en la entrada hay la información de m asignaturas
// Post: retorna un vector de Asignatura con la información de la entrada
//          manteniendo el orden
vector <Asignatura> lee_asignaturas(int m) {
    vector <Asignatura> asig(m);
    for (int i = 0; i < m; ++i) cin >> asig[i].id >> asig[i].creditos >> asig[i].nota;
    return asig;
}

int main() {
    int n;
    cin >> n;
    Alumnos a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i].dni;
        int m;
        cin >> m;
        a[i].asigs = lee_asignaturas(m);
    }
    cout << supera_alfa(a) << endl;
}
```

El salto del caballo

Opción 1: Solución compacta que aprovecha la observación final del enunciado para caracterizar los movimientos del caballo. Existen otras soluciones que aprovechan la observación final del enunciado.

```
// Pre: -
// Post: retorna el valor absoluto
int abs(int a) {
    if (a < 0) return -a;
    return a;
}

// Pre: m * n es el tamanno de un tablero
// Post: retorna true si (i,j) es una posicion válida del tablero y false en caso contrario
bool es_valida(int m, int n, int i, int j) {
    return 0 <= i and i < m and 0 <= j and  j < n;
}

// Pre: tab describe los posibles movimientos de un caballo; (i,j) es una
//     posición válida de tab; testigo tiene un valor válido
// Post: actualiza tab asignando value a las posiciones libres de tab (las de valor 0)
//      que son accesibles desde la posición (i,j). El valor de testigo es true cuando ese
//      era su valor al inicio o ha habido algún cambio en tab
void move_update_from(Tablero& tab, int value, int i, int j, bool& testigo) {
    int m = tab.size();
    int n = tab[0].size();
    for (int a = -2;  a < 3; ++a)
        for (int b = -2; b < 3; ++b)
            if (abs(a) + abs(b) == 3 and es_valida(m, n, i + a, j + b)
                                     and tab[i + a][j + b] == 0) {
                tab[i + a][j + b] = value;
                testigo = true;
            }
}

// Pre: tab es la configuración del tablero cuando el caballo ha hecho k-1 saltos y
//      k >= 1 indica que se ha de simular el k-ésimo salto.
// Post: tab es la configuración del tablero cuando se ha hecho el k-ésimo salto;
//      testigo es true si tab ha cambiado y false en caso contrario
void move_update(Tablero& tab, int k, bool& testigo) {
    testigo = false;
    int m = tab.size();
    int n = tab[0].size();
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < n; ++j)
            if (tab[i][j] == k) move_update_from(tab, k + 1, i, j, testigo);
}
```

Opción 2: Solución más básica (y menos preferible) con intrucciones condicionales de exploración del caballo.

```
// Pre: i, j son índices válidos de tab; b es válido
// Post: tab[i][j] actualiza su valor a v si era 0 y b se actualiza a true;
//       en cualquier otro caso no se producen cambios
void move(Tablero& tab, int v, int i, int j, bool& b) {
  if (tab[i][j] == 0) {
    tab[i][j] = v;
    b = true;
  }
}

// Pre: tab describe los posibles movimientos de un caballo; (i,j) es una
//     posición válida de tab; testigo tiene un valor válido
// Post: actualiza tab asignando value a las posiciones libres de tab (las de valor 0)
//      que son accesibles desde la posición (i,j). El valor de testigo es true cuando ese
//      era su valor al inicio o ha habido algún cambio en tab
void move_update_from(Tablero& tab, int value, int i, int j, bool& testigo) {
  int m = tab.size();
  int n = tab[0].size();

  bool jplus = j + 1 < n;
  bool jminus = j - 1 >= 0;
  if (i + 2 < m) {
    if (jplus) move(tab, value, i + 2, j + 1, testigo);
    if (jminus) move(tab, value, i + 2, j - 1, testigo);
  }
  if (i - 2 >= 0) {
    if (jplus) move(tab, value, i - 2, j + 1, testigo);
    if (jminus) move(tab, value, i - 2, j - 1, testigo);
  }

  bool iplus = i + 1 < m;
  bool iminus = i - 1 >= 0;
  if (j + 2 < n) {
    if (iplus) move(tab, value, i + 1, j + 2, testigo);
    if (iminus) move(tab, value, i - 1, j + 2, testigo);
  }
  if (j - 2 >= 0) {
    if (iplus) move(tab, value, i + 1, j - 2, testigo);
    if (iminus) move(tab, value, i - 1, j - 2, testigo);
  }
}

// Pre: tab es la configuración del tablero cuando el caballo ha hecho k-1 saltos y
//      k >= 1 indica que se ha de simular el k-ésimo salto.
// Post: tab es la configuración del tablero cuando se ha hecho el k-ésimo salto;
//      testigo es true si tab ha cambiado y false en caso contrario
void move_update(Tablero& tab, int k, bool& testigo) {
  testigo = false;
  int m = tab.size();
  int n = tab[0].size();
  for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j)
      if (tab[i][j] == k) move_update_from(tab, k + 1, i, j, testigo);
```

```
}
```