```
// HEADERS ------------------------------------------------------------

#include <string.h>
#include <fstream>
#include <iostream>
using namespace std;

// STRUTTURE DATI -----------------------------------------------------
struct Record {
  char name[128];
  int value;
};
struct RecordSet {
  Record* records;
  int size;
  int capacity;
};
// FUNZIONI -----------------------------------------------------------
-------
RecordSet init() {
  RecordSet rs;
  rs.records = nullptr;
  rs.size = 0;
  rs.capacity = 0;
  return rs;
}
void drop(RecordSet& rs) {
  if (rs.records != nullptr) {
    delete[] rs.records;
    rs.records = nullptr;
    rs.size = 0;
    rs.capacity = 0;
  }
}
int search(RecordSet& rs, const char* name);
int insert(RecordSet& rs, const char* name, int value) {
  if (search(rs, name) != -1) {
    return -1;
  }
  if (rs.size == rs.capacity) {
    Record* tmp = new Record[rs.capacity * 2];
    for (int i = 0; i < rs.size; i++) {
      tmp[i] = rs.records[i];
    }
    delete[] rs.records;
    rs.records = tmp;
    rs.capacity *= 2;
  }
  strcpy(rs.records[rs.size].name, name);
  rs.records[rs.size].value = value;
  rs.size += 1;
  return rs.size - 1;
}
int search(RecordSet& rs, const char* name) {
  for (int i = 0; i < rs.size; i++) {
    if (strcmp(rs.records[i].name, name) == 0) {
      return i;
    }
  }
  return -1;
```

```
}
int remove(RecordSet& rs, const char* name) {
  int pos = search(rs, name);
  if (pos == -1) {
    return -1;
  }
  for (int i = pos; i < rs.size - 1; i++) {
    rs.records[i] = rs.records[i + 1];
  }
  rs.size--;
  return pos;
}
int update(RecordSet& rs, const char* name, int value) {
  int pos = search(rs, name);
  if (pos == -1) {
    return -1;
  }
  rs.records[pos].value = value;
  return pos;
}
int load(RecordSet& rs, const char* filename) {
  ifstream ifs(filename);
  if (!ifs) {
    return -1;
  }
  drop(rs);
  ifs >> rs.size;
  rs.capacity = rs.size;
  rs.records = new Record[rs.size];
  for (int i = 0; i < rs.size; i++) {
    ifs >> rs.records[i].name >> rs.records[i].value;
  }
  return rs.size;
}
int save(RecordSet& rs, const char* filename) {
  ofstream ofs(filename);
  if (!ofs) {
    return -1;
  }
  ofs << rs.size << endl;
  for (int i = 0; i < rs.size; i++) {
    ofs << rs.records[i].name << " " << rs.records[i].value << endl;
  }
  return rs.size;
}
#ifndef __TESTS__
int main() {
  // da usare per i vostri tests
  return 0;
}
#endif
```

PROVA 2 con matrici

```
// HEADERS ---------------------------------------------------------------

#include <string.h>
#include <fstream>
#include <iostream>
using namespace std;

// STRUTTURE DATI --------------------------------------------------------

struct GameWorld {
  int* cells;
  int rows;
  int columns;
};

// FUNZIONI --------------------------------------------------------------

int get(const GameWorld& gw, int i, int j) {
  if (i < 0 || i >= gw.rows || j < 0 || j >= gw.columns) return 0;
  return gw.cells[i * gw.columns + j];
}
void set(GameWorld& gw, int i, int j, int value) {
  if (i < 0 || i >= gw.rows || j < 0 || j >= gw.columns) return;
  gw.cells[i * gw.columns + j] = value;
}
int count(GameWorld& gw, int i, int j) {
  int count = 0;
  for (int k = i - 1; k <= i + 1; k++)
    for (int l = j - 1; l <= j + 1; l++)
      if (k != i || l != j) count += get(gw, k, l);
  return count;
}
GameWorld copy(const GameWorld& gw) {
  GameWorld gw2;
  gw2.rows = gw.rows;
  gw2.columns = gw.columns;
  gw2.cells = new int[gw.rows * gw.columns];
  for (int i = 0; i < gw.rows * gw.columns; i++) gw2.cells[i] =
gw.cells[i];
  return gw2;
}
GameWorld init(int n, int m, int* cells) {
  GameWorld gw;
  gw.rows = n;
  gw.columns = m;
  gw.cells = new int[n * m];
  if (cells != nullptr) {
    for (int i = 0; i < n * m; i++) gw.cells[i] = cells[i];
  } else {
    for (int i = 0; i < n * m; i++) gw.cells[i] = 0;
  }
  return gw;
}
void drop(GameWorld& gw) {
  delete[] gw.cells;
  gw.cells = nullptr;
  gw.rows = 0;
```

```
  gw.columns = 0;
}
int step(GameWorld& gw) {
  GameWorld gw2 = copy(gw);
  int changes = 0;
  for (int i = 0; i < gw.rows; i++) {
    for (int j = 0; j < gw.columns; j++) {
      int c = count(gw, i, j);
      int v = get(gw, i, j);
      if (v == 0 && c == 3) {
        set(gw2, i, j, 1);
        changes++;
      } else if (v == 1 && (c < 2 || c > 3)) {
        set(gw2, i, j, 0);
        changes++;
      }
    }
  }
  drop(gw);
  gw = gw2;
  return changes;
}
GameWorld load(const char* filename) {
  ifstream ifs(filename);
  if (!ifs) return init(0, 0, nullptr);
  int n, m;
  ifs >> n >> m;
  int* cells = new int[n * m];
  for (int i = 0; i < n * m; i++) ifs >> cells[i];
  GameWorld gw = init(n, m, cells);
  delete[] cells;
  return gw;
}
int save(const GameWorld& gw, const char* filename) {
  ofstream ofs(filename);
  if (!ofs) return -1;
  ofs << gw.rows << " " << gw.columns << endl;
  for (int i = 0; i < gw.rows; i++) {
    for (int j = 0; j < gw.columns; j++) {
      ofs << get(gw, i, j) << " ";
    }
    ofs << endl;
  }
  return 0;
}
void print(const GameWorld& gw) {
  printf("GameWorld %d x %d\n", gw.rows, gw.columns);
  for (int i = 0; i < gw.rows; i++) {
    for (int j = 0; j < gw.columns; j++) {
      printf("%d ", get(gw, i, j));
    }
    printf("\n");
  }
  printf("\n");
}
#ifndef __TESTS__
int main() {
  // da usare per i vostri tests
  return 0;
}#endif
```

Prova 3

```
// HEADERS ---------------------------------------------------------------
#include <string.h>
#include <fstream>
#include <iostream>
using namespace std;
// STRUTTURE DATI --------------------------------------------------------
struct Record {
  char* name;
  int grade;
};
struct RecordSet {
  Record* records;
  int size;
};
// FUNZIONI --------------------------------------------------------------
RecordSet init(int size) {
  if (size == 0) return {nullptr, 0};
  RecordSet rs = {new Record[size], size};
  for (int idx = 0; idx < rs.size; idx++) {
    rs.records[idx].name = nullptr;
    rs.records[idx].grade = -2;
  }
  return rs;
}
void drop(RecordSet& rs) {
  for (int idx = 0; idx < rs.size; idx++) {
    delete[] rs.records[idx].name;
  }
  delete[] rs.records;
  rs.records = nullptr;
  rs.size = 0;
}
int insert(RecordSet& rs, const char* name, int grade) {
  for (int idx = 0; idx < rs.size; idx++) {
    if (rs.records[idx].name == nullptr) {
      rs.records[idx].name = new char[strlen(name) + 1];
      strcpy(rs.records[idx].name, name);
      rs.records[idx].grade = grade;
      return idx;
    }
  }
  return -1;
}




int search(const RecordSet& rs, const char* name) {
  for (int idx = 0; idx < rs.size; idx++) {
    if (rs.records[idx].name == nullptr) continue;
    if (strcmp(rs.records[idx].name, name) == 0) return idx;
  }
  return -1;
}
int update(RecordSet& rs, const char* name, int grade) {
  int pos = search(rs, name);
  if (pos == -1) return -1;
  rs.records[pos].grade = grade;
```

```
  return pos;
}
int remove(RecordSet& rs, const char* name) {
  int pos = search(rs, name);
  if (pos == -1) return -1;
  delete[] rs.records[pos].name;
  rs.records[pos].name = nullptr;
  rs.records[pos].grade = -2;
  return pos;
}
RecordSet load(const char* filename) {
  ifstream fs(filename);
  if (!fs) return init(0);
  int num;
  fs >> num;
  RecordSet rs = init(num);
  char buf[256];
  int grade;
  for (int idx = 0; idx < rs.size; idx++) {
    fs >> buf >> grade;
    if (strcmp(buf, "CANCELLATO") == 0) {
      rs.records[idx].name = 0;
      rs.records[idx].grade = -2;
    } else {
      rs.records[idx].name = new char[strlen(buf) + 1];
      strcpy(rs.records[idx].name, buf);
      rs.records[idx].grade = grade;
    }
  }
  return rs;
}
int save(const RecordSet& rs, const char* filename) {
  ofstream fs(filename);
  if (!fs) return -1;
  fs << rs.size << endl;
  for (int idx = 0; idx < rs.size; idx++) {
    fs << (rs.records[idx].name ? rs.records[idx].name : "CANCELLATO") <<
" "
       << rs.records[idx].grade << endl;
  }
  return 0;
}
#ifndef __TESTS__
int main() {
  // da usare per i vostri tests
  return 0;
}
#endif
```