1 Permutation Flowshop Problem

```
void readInstance(const std::string &instanceFile) {
  std::string line, tmp;

  std::ifstream input(instanceFile);
  if (!input)
  input >> N >> M >> seed;
  due_date.resize(N);
  processing_time.resize(M);
  for(unsigned long long int i = 0; i < M; i++){
    processing_time[i].resize(N);
  }

  total_job_processing_time.resize(N);
  std::fill(total_job_processing_time.begin(), total_job_processing_time.end(), 0);
  int num_job;
  total_processing_time = 0;
  for (unsigned long long int i = 0; i < N; i++) {
    input >> num_job >> due_date[i];
    for(unsigned long long int j = 0; j < M; j++){
      input >> processing_time[j][i];
      total_processing_time += processing_time[j][i];
      total_job_processing_time[i] += processing_time[j][i];
    }
  }
}
```

2 Travelling Salesman Problem

```
   void readInstance(const std::string &instanceFile) {
std::string line, tmp;

std::ifstream input(instanceFile);
if (!input)
  throw std::runtime_error("Error: unable to open benchmark file");

//Get Dimension
while (line.rfind("DIMENSION", 0) != 0)
  std::getline(input, line);

std::vector<std::string> data = core::StringHelper::get()->split(line, " ");
numberOfCities =
static_cast<unsigned long long int>(std::stoi(data[data.size() - 1]));


//Get Edge_weight_type
while (line.rfind("EDGE_WEIGHT_TYPE", 0) != 0)
  std::getline(input, line);

data = core::StringHelper::get()->split(line, " ");
tmp = data[data.size() - 1];

//Parse
if (tmp.rfind("EXPLICIT", 0) == 0) {
  distanceMatrix.resize(numberOfCities);
  // skip file info
  while (line.rfind("START", 0) != 0)
    std::getline(input, line);

  for (unsigned long long int i = 0; i < numberOfCities; i++) {
    distanceMatrix[i].resize(numberOfCities);
    for (unsigned long long int j = 0; j < numberOfCities; j++) {
      input >> distanceMatrix[i][j];
    }
  }

  while (line.rfind("EOF", 0) != 0)
    std::getline(input, line);

} else if (tmp.rfind("EUC_2D", 0) == 0) {

  while (line.rfind("NODE_COORD_SECTION", 0) != 0)
    std::getline(input, line);
    cities.resize(numberOfCities);
    for (unsigned long long int i = 0; i < numberOfCities; i++) {
      input >> cities[i].id >> cities[i].x >> cities[i].y;
    }
    computeDistances();
  } else {
    std::cerr << "INVALID FORMAT" << std::endl;
  }
}

/**
 * Compute the distance matrix
 */
void computeDistances() {
  distanceMatrix.resize(numberOfCities);
  for (unsigned long long int i = 0; i < numberOfCities; i++) {
    distanceMatrix[i].resize(numberOfCities);
    distanceMatrix[i][i] = 0;
    for (unsigned long long int j = 0; j < i; j++) {
      if (i != j) {
        double x = sqrt(std::pow((cities[i].x - cities[j].x), 2) +
        std::pow((cities[i].y - cities[j].y), 2)) + 0.5;
        distanceMatrix[i][j] = (int) x;
        distanceMatrix[j][i] = (int) x;
      }
    }
  }
}

double getDistance(unsigned long long int i, unsigned long long int j){
  return distanceMatrix[i][j];
}

unsigned long long int getMatrixSize(){
  return distanceMatrix.size();
}

unsigned long long int getNumberOfCities(){
  return numberOfCities;
}

void generate(int _n, int _seed, int ub=3163) {
  numberOfCities = _n;
  seed = _seed;
  if (_seed != -1)
    core::RNGHelper::get()->reseed(_seed);

  cities.resize(numberOfCities);

  for (unsigned long long int i=0; i < numberOfCities; i++) {
    cities[i].id = i;
    cities[i].x = core::RNGHelper::get()->uniform(ub);
    cities[i].y = core::RNGHelper::get()->uniform(ub);
  }

  computeDistances();
}
}
```