```
template<class CELL, class FACE, class NODE>
inline void INPUT_GRID<CELL, FACE, NODE>::SetNumberObjects(counter_t&
number_cells, counter_t& number_faces, counter_t& number_nodes,
counter_t& number_ghosts, counter_t& number_groups)
{
      in >> number_cells_;
      in >> number_faces_;
      in >> number_nodes_;
      in >> number_groups_;

      sizes_.reserve(number_groups_);
      counter_t value;
      for(std::size_t  i=0; i< number_groups_; ++i){
      in >> value;
      sizes_.push_back(value);
      };

      number_ghosts_ =std::accumulate(sizes_.begin(), sizes_.end(), 0);

      number_cells =number_cells_;
      number_faces=number_faces_;
      number_nodes=number_nodes_;
      number_ghosts=number_ghosts_;
      number_groups=number_groups_;
return;};

template<class CELL, class FACE, class NODE>
inline  void INPUT_GRID<CELL, FACE, NODE>::ReadCells(unordered_set<  CELL
>& cells)
{
      typename CELL::AdjacentNodes cell_adjacent_nodes;
      typename CELL::AdjacentFaces cell_adjacent_faces;
      CELL* c;
      for(std::size_t idx_cell=0; idx_cell<number_cells_; ++idx_cell){
      c = new CELL( idx_cell+1 );
      in >> geotype_;
      c-> set_geometry(geotype_);
      for(std::size_t i=0; i < cell_adjacent_faces.size(); ++i)     in >>
cell_adjacent_faces[i];
      for(std::size_t i=0; i < cell_adjacent_nodes.size(); ++i)     in >>
cell_adjacent_nodes[i];
      c -> set_adjacent_faces(cell_adjacent_faces);
      c -> set_adjacent_nodes(cell_adjacent_nodes);
      cells.insert(*c);
//    c->print();
      };
return;
};

template<class CELL, class FACE, class NODE>
inline  void INPUT_GRID<CELL, FACE, NODE>::ReadFaces(unordered_set<  FACE
>& faces)
{
      typename FACE::AdjacentNodes face_adjacent_nodes;
      typename FACE::AdjacentCells face_adjacent_cells;
      FACE* f;
      for(std::size_t idx_face=0; idx_face<number_faces_;++idx_face ){
      f = new FACE (idx_face+1);
      in>>geotype_;
      f ->set_geometry( geotype_ );
```

```
      for(std::size_t i=0; i<face_adjacent_nodes.size(); ++i)
in>>face_adjacent_nodes[i];
      for(std::size_t i=0; i<face_adjacent_cells.size(); ++i)
in>>face_adjacent_cells[i];
      f-> set_adjacent_cells( face_adjacent_cells );
      f-> set_adjacent_nodes( face_adjacent_nodes );
      faces.insert(*f);
//    f->print();
      };
return;
};

template<class CELL, class FACE, class NODE>
inline  void INPUT_GRID<CELL, FACE, NODE>::ReadNodes(unordered_set<  NODE
>& nodes)
{
      typename NODE::coords_t xyz;
      NODE* n;
      for(std::size_t idx_node=0; idx_node<number_nodes_;++idx_node ){
      n = new NODE(idx_node+1);
      for(std::size_t i=0; i<xyz.size(); ++i)in>>xyz[i];
      n -> set_cartesian_coordinates(xyz);
      nodes.insert(*n);
//    n->print();
      };
return;
};

template<class CELL, class FACE, class NODE>
inline  void INPUT_GRID<CELL, FACE, NODE>::ReadBoundary(boundary_t&
boundary)
{
      boundary.reserve(number_groups_);
      for(typename std::vector<counter_t>::iterator
p=sizes_.begin();p!=sizes_.end(); ++p)
      {
      std::valarray<counter_t> arr(*p);
      for(counter_t* q=valarray_begin<counter_t>(arr);
q!=valarray_end<counter_t>(arr); ++q) in>>*q;
      boundary.push_back(arr);
      };
return;
};
```