Summary of the template class library

Enumerators

Template parameter *Item* = datatype of the enumerated elements.

Type	**Description**
ArrayEnumerator<Item>	The constructor needs the pointer of the array to be enumerated
SeqInFileEnumerator<Item>	The constructor needs the name of the file to be enumerated. **It has** one **exception**. If *Item* is a complex structure, reading operator has to be defined for that.
StringStreamEnumerator<Item>	The constructor needs the stringstream object to be enumerated. If *Item* is a complex structure, reading operator has to be defined for that.
IntervalEnumerator	The constructor needs the upper and the lower bound of the interval. It enumerates only integers, template parameter is not needed.

Algorithmic patterns

Common public methods:

addEnumerator(): it connects the enumerator to the pattern
  Input: the reference or the pointer of the enumerator
  Output: -

run(): it runs the pattern
  Input: -
  Output: -

Common protected methods (to override):

first(): it calls method *first()* of the enumerator
  Input: -
  Output: -

whileCond(): should not be overridden in the children of Selection. It is used to implement the *as long as* condition.
  Input: current element of the enumerator
  Output: a condition. The algorithmic pattern is run as long as its basic loop condition and this condition hold.

The patterns have one **exception** called MISSING_ENUMERATOR which is thrown when the enumerator is not connected to the pattern.

Template parameter *Item* means the datatype of the enumerated elements.

Type	**Description**
Summation<Item,Value=Item>	Template parameter *Value* means the datatype of the result. Default value: Item To be overridden: **func():** *Input:* current element of the enumerator *Output:* the value which can be added to the result of the summation **neutral():** *Input:* - *Output:* Initial value of the result **add():** *Input:* two variables of type Value *Output:* how the two variables are added to each other **cond():** *Input:* current element of the enumerator *Output:* the condition of the summation. If it is satisfied, the current element is added to the result. Default value: true (for simple summation). Getter: **result():** gives the result of the Summation
Summation<Item,vector<Value> >	Template parameter *Value* means the datatype of the elements of the result vector. To be overridden **func():** *Input:* current element of the enumerator *Output:* the value which can be concatenated to the result vector **cond():** *Input:* current element of the enumerator *Output:* the condition of the summation. If it is satisfied, the current element is concatenated to the result vector. Default value: true (for simple summation). Getter: **result():** gives the result vector
Summation<Item,ostream>	It is used to write to the **console** or to a **file**. To be overridden: **func():** *Input:* current element of the enumerator *Output:* the string which can be written to the ostream (file/console). If data is written to a file, the file has to be opened before (e.g. in the constructor), and closed after using it (e.g. in the destructor). **cond():** *Input:* current element of the enumerator *Output:* the condition of the summation. If it is satisfied, the current element is added to the result. Default value: true (for simple summation). Getters: not needed, the result can be seen in a file or in the console

Counting<Item>	**To be overridden:** **cond():** *Input:* current element of the enumerator *Output:* the condition of the counting. If it is satisfied, the result of the counting is increased. Default value: true **Getter:** **result():** gives the result of the Counting
LinSearch<Item,bool >	The bool template parameter determines if it is an optimistic (true) or a pessimistic (false) linear search. **To be overridden:** **cond():** *Input:* current element of the enumerator *Output:* the condition of the linear search. If it is satisfied, the pessimistic search stops. If it is not satisfied, the optimistic search stops. **Getters:** **found():** getter of the bool value which indicates if the search was successful **elem():** getter of the searched element
MaxSearch<Item,Value, Compare>	Template parameter *Value* means the datatype of the values to be compared in the maximum search. Template parameter *Compare* is a datatype which indicates if it is a maximum (Greater<Value>) or a minimum (Less<Value>) search. **To be overridden:** **func():** *Input:* current element of the enumerator *Output:* the value which can be compared to find the maximum **cond():** *Input:* current element of the enumerator *Output:* the condition of the maximum search. If it is satisfied, the current element is compared with the others. Default value: true (for simple maximum selection). **Getters:** **found():** returns if there was at least one element satisfying the condition **opt():** gives the maximum value **optElem():** gives the element for which the maximum is got
Selection<Item>	**To be overridden:** **cond():** *Input:* current element of the enumerator *Output:* the condition of the selection. If it is satisfied, the selection stops. **Getter:** **result():** gives the element for which the condition holds