Classes reference

1. Project structure – *TuringMachine*
   * *Header Files* – contains only C++ *.h* files
     + *Abstractions* – contains the abstract classes of the application
     + *Declarations –* contains class declarations
       - *IO –* stores the declarations of all input-output related classes
       - *Machine –* stores the declarations of all machine related classes including the machine strategies
       - *Utilities –* the helper classes are declared here
     + *Structures –* the implementations of the data structure templates are stored here
   * *Source Files –* all *.cpp* files are stored here
     + *Definitions –* all classes declared in *.h* files are defined symmetrically here
       - *IO –* the IO classes definitions are here
       - *Machine –* stores the machine classes definitions
       - *Utilities –* the helper classes are defined here
     + *ProgramEntry.cpp –* the application entry point which contains the *main* method
2. Turing machine implementation related classes
   * *TuringMachine –* implements the mathematics model of the machine
     + *Type definitions – string* by default, the developer should change them if wants to use custom types
       - *StatesType –* defines which type for states will be used
       - *SymbolType –* defines which type for alphabet characters will be used
       - *TransitionKeyType –* defines the type which will be used as a request argument in the transition function
       - *TransitionKeyValue –* defines the type which will be used as a return value from the transition function
     + *Data structures type definitions –* defines which types will be used to store the machine elements, the developer should change them if wants to use custom types
       - *AlphabetSet*
       - *StatesSet*
       - *FinalStatesSet*
       - *TransitionF*
       - *MachineTape*
     + *Constructor –* creates empty data structures
     + *Get-ers for the machine data structures*
     + *Get-er / set-er for the start state*
     + *Get-er / set-er for the current state*
   * *MachineStrategy –* abstract class for implementing family of algorithms according to the Strategy design pattern.
     + *Calculate(TuringMachine\*) –* to start the algorithm call this method with the machine instance as an argument. The algorithm can use the machine interface to interact with its state.
   * *RunStartToEndStrategy –* implementation of the *MachineStrategy.* Used by default. The algorithm gets the start configuration and calls recursively the transition function until a final state is reached or the transition function is undefined with given state
3. Input-output related classes
   * *IOServiceFactory –* contains static factory method for creating *IOService* objects
     + *createService() –* this method reads the first two lines of the input and decides which type of IO stream will be used
   * *IOService –* abstract class which declares methods for working with streams. The developer should derive from it to implement a certain stream functionality
     + *readMachineInput(TuringMachine&) –* parses information from the input stream and stores it in the machine data structures
     + *writeMachineOutput(TuringMachine&) –* converts the machine’s final configuration and writes it in the output stream
   * *FileIOService –* derives from *IOService -* provides methods for working with file stream
   * *StandartIOService –* derives from IOService – provides methods for working with console stream
4. Data structure abstract classes
   * *Set<T> –* abstract template class which declares the interface of a set data structure
     + *void add(T) –* method for adding elements to the set
     + *bool contains(T&) –* returns *true* if the set contains the given value
   * *Tape<T> -* abstract class which declares the interface of the data structure which represents the machine tape
     + *void add(T) ­*– adds element in the ‘right’ end of the tape
     + *get / setPosition(T) –* gets or sets the value of the current position
     + *void move(string) ­–* accepts ‘R’, ‘L’ or ‘S’ as argument and move the current head position in the respective direction
     + *void moveLeft() –* moves the head one position in the left
     + *void moveRight() –* moves the head one position in the right
     + *get / setBlank() –* gets or sets the blank symbol of the alphabet
     + *void trimLeft() –* the blank symbols in the left end of the tape
     + *void trimRight() –* removes the blank symbols in the right end of the tape
     + *void trim() –* removes the blank symbols in the both ends
     + *string toString() –* returns the string representation of the machine configuration
   * *TransitionFunction<T, V> -* abstract class which declares the interface for working with the data structure representing the machine’s transition function
     + *void addRule(T,V) –* adds a transition entry where the *T*  argument is an object representing the combination of the requested state and the alphabet symbol and the *V* argument is the object that will be returned with this request
     + *V transite(T) –* returns an object which represent combination of a state, alphabet symbol and direction. This object is correlated to the request
5. Data structures implementation
   * *DLLTape : public Tape<T> -* this is double linked list implementation of the tape. It represents the sequence between the fist and the last not blank symbols in the tape.
   * *StlSet : public Set<T> -* implementation of the *Set* interface based on the *stl::set<T>* data structure
   * *StlTransitionFunction : public TransitionFunction<T,V> -* implementation of the *TransitionFunction* interface base on the *stl::map<T,V>* data structure
6. Helper classes
   * *CommonMethods –* provides methods for executing simple tasks
     + *bool isEmptyFilePath(string) –* returns true if the string is empty or ‘-’
     + *void splitTransitionRule(string, string&, string&, string&, string&, string&) –* the first parameter is the string representation of a transition function entry and the method split it in the provided variables
     + *void splitTransitionResult(string, string&, string&, string&) –* the first parameter is the string representation of a transition function response and it is split in the provided variables