```cpp
#include <iomanip>
#include <iostream>

#include "../../!includes/State_Machine/command_line_table.h"
#include "../../!includes/Enums/enumerations.h"


using namespace  std;

//============================================================
///      FUNCTION DEFINITIONS
//============================================================

void initialize_cmd( int table[ ][ CMD_COL ], int row, int col )
{
    ///initializing array elements to -1
    for ( int i = 0; i < row; i++ )
    {
        for ( int j = 0; j < col; j++ )
        {
            table[ i ][ j ] = static_cast<int>( states::start );
            /// initializing array elements to -1 and
            set_fail( table, i);
            /// set col 0 to state 0 / fail state
        }
    }
}

void mark_cells( int table[ ][ CMD_COL ], int row, cmd_range r,
int state )
{
    for( char col = r.start ; col <= r.end; col++)
    {
        /// mark cells with cmd_range to this state
        table[ row ][ int( col ) ] = state;
    }
}

void mark_cells( int table[][CMD_COL], int row, int col, int
state )
{
    /// mark cells at row and col with this state
    table[ row ][ col ] = state;
}
```

```
void mark_cells( int table[][CMD_COL], int row, char column[],
int state)
{
    for( size_t i = 0  ; i < strlen(column); i++)
    {
        // based on their ascii values, mark array at
        // set table at column 0 and row = row to 1 / success
        char temp = column[ i ];
        table[ row ][ int ( temp ) ] = state;
    }
}

void set_success( int table[][CMD_COL], unsigned int row )
{
   table[row][0] = static_cast<int>( states::success );
   // set table at column 0 and row = row to 1 / success
}

void set_fail(int table[][CMD_COL], int row)
{
   table[ row ][ 0 ] = static_cast<int>( states::fail );
   // set table at column 0 and row = row to 0 / fail
}

void fill_table( int table[ ][ CMD_COL ] )
{
    /// \note start states are select, Make , Batch,...

    fill_create_machine( table );
    fill_drop_machine  ( table );
    fill_delete_machine( table );
    fill_make_machine  ( table ) ;
    fill_select_machine( table );
    fill_insert_machine( table );

    /// set sql select statement with different syntax
    /// eg. select * from employee.bin
    ///     select last, first , age from empoyee
    ///     select last, first , age from empoyee where age >=23

}

//
================================================================
==
// modifiers for each state
//
```

```
==================================================================
==

void fill_create_machine( int table[ ][ CMD_COL ] )
{
    ///// mark_cells( table, row, column, state )
    //
==================================================================
==
    // create table employee fields  last, first, dep, salary,
year
    mark_cells( table , 0   ,
static_cast<int>( KEYWORDS::CREATE) , 1   );
    /// create
    mark_cells( table , 1   ,
static_cast<int>( KEYWORDS::TABLE) , 21   );
    /// table
    mark_cells( table , 21 ,
static_cast<int>( KEYWORDS::SYMBOL) , 4   );
    /// 'employee'
    mark_cells( table , 4   ,
static_cast<int>( KEYWORDS::FIELDS) , 22   );
    /// fields
    mark_cells( table , 22  ,
static_cast<int>( KEYWORDS::SYMBOL) , 2   );
    /// last
    mark_cells( table , 2   ,
static_cast<int>( KEYWORDS::SYMBOL) , 9 );
    /// for commas
    mark_cells( table , 9 ,
static_cast<int>( KEYWORDS::SYMBOL) , 2   );
    /// fields after comman ( Age )
    set_success( table , 2 );

}

void fill_delete_machine ( int table[ ][ CMD_COL ] )
{
    ///// mark_cells( table, row, column, state )
    //
==================================================================
==
    // delete from employee where Last = "Jack"
    mark_cells( table , 0   ,
static_cast<int>( KEYWORDS::DELETE ) , 1   );
    /// delete
    mark_cells( table , 1   ,
static_cast<int>( KEYWORDS::FROM ) , 3  );
    /// from
    mark_cells( table , 3   ,
static_cast<int>( KEYWORDS::SYMBOL ) , 4 );
    /// employee
    mark_cells( table , 4   ,
static_cast<int>( KEYWORDS::WHERE ) , 7 );

    mark_cells( table , 7   ,
static_cast<int>( KEYWORDS::SYMBOL ) , 6 );
    /// (
    mark_cells( table , 6   ,
static_cast<int>( KEYWORDS::SYMBOL ) , 6 );
    set_success( table , 6 );

}

void fill_drop_machine(  int table[ ][ CMD_COL] )
{
    ///// mark_cells( table, row, column, state )
    //
=================================================================
==
    // drop table employee
    mark_cells( table , 0   , static_cast<int>( KEYWORDS::DROP) ,
41  );
    /// drop
    mark_cells( table , 41   ,
static_cast<int>( KEYWORDS::TABLE) , 31  );
    /// table
    mark_cells( table , 31 ,
static_cast<int>( KEYWORDS::SYMBOL) , 34  );
    /// 'employee'
    set_success( table , 34 );
}

void fill_make_machine(  int table[ ][ CMD_COL] )
{
    mark_cells( table , 0   , static_cast<int>( KEYWORDS::MAKE) ,
1  );
    /// create
    mark_cells( table , 1   ,
static_cast<int>( KEYWORDS::TABLE) , 21  );
    /// table
    mark_cells( table , 21 ,
static_cast<int>( KEYWORDS::SYMBOL) , 4  );
    /// 'employee'
    mark_cells( table , 4   ,
static_cast<int>( KEYWORDS::FIELDS) , 22   );
    /// fields
    mark_cells( table , 22   ,
static_cast<int>( KEYWORDS::SYMBOL) , 2   );
    /// last
    mark_cells( table , 2   ,
static_cast<int>( KEYWORDS::SYMBOL) , 9 );
    /// for commas
    mark_cells( table , 9 ,
static_cast<int>( KEYWORDS::SYMBOL) , 2   );
    /// fields after comman ( Age )
    set_success( table , 2 );
}

void fill_select_machine( int table[ ][ CMD_COL ] )
{
    ///// mark_cells( table, row, column, state )
    //
=================================================================
==
    // select * from Table
    mark_cells( table , 0   ,
static_cast<int>( KEYWORDS::SELECT) , 1   );
    mark_cells( table , 1   , static_cast<int>( KEYWORDS::STAR) ,
2   );
    mark_cells( table , 2   , static_cast<int>( KEYWORDS::FROM) ,
3   );
    mark_cells( table , 3   ,
static_cast<int>( KEYWORDS::SYMBOL) , 4 );
    set_success( table , 4 );

    //
=================================================================
==
    // select * from Table where ( Age < 27 && Last = Tom )
    mark_cells( table , 4   ,
static_cast<int>( KEYWORDS::WHERE) , 7 );
    mark_cells( table , 7   ,
static_cast<int>( KEYWORDS::SYMBOL) , 6 );
    /// (
    mark_cells( table , 6   ,
static_cast<int>( KEYWORDS::SYMBOL) , 6 );
    set_success( table , 6 );

    //
=================================================================
==
    // select Last, First, Age from Table
    mark_cells( table , 1   ,
static_cast<int>( KEYWORDS::SYMBOL) , 2 );
    /// fields ( last first ...
    mark_cells( table , 2   ,
static_cast<int>( KEYWORDS::SYMBOL) , 9 );
    /// for commas
    mark_cells( table , 9 ,
static_cast<int>( KEYWORDS::SYMBOL) , 2  );
    /// fields after comman ( Age )
    mark_cells( table , 2   , static_cast<int>( KEYWORDS::FROM) ,
3  );
    /// from table

}

void fill_insert_machine( int table[ ][ CMD_COL ]  )
{
    ///// mark_cells( table, row, column, state )
    //
=================================================================
==
    // insert into Table values  Tom, Joe, CS, 100000, 2018

//    mark_cells( table , 0   ,
static_cast<int>( KEYWORDS::INSERT) , 1  );
//    /// insert
//    mark_cells( table , 1   ,
static_cast<int>( KEYWORDS::INTO) , 20 );
//    /// into
//    mark_cells( table , 20 ,
static_cast<int>( KEYWORDS::SYMBOL) , 4  );
//    /// 'table'
//    mark_cells( table , 4   ,
static_cast<int>( KEYWORDS::SYMBOL) , 19 );
//    /// values
//    mark_cells( table , 19 ,
static_cast<int>( KEYWORDS::SYMBOL) , 11  );
//    /// "Tom"
//    mark_cells( table , 11  ,
static_cast<int>( KEYWORDS::SYMBOL) , 19 );
//    /// for commas
//    set_success( table, 11 );

    mark_cells( table , 0   ,
static_cast<int>( KEYWORDS::INSERT) , 1  );
```

```
    /// insert
    mark_cells( table , 1  , static_cast<int>( KEYWORDS::INTO) ,
13 );
    /// into
    mark_cells( table , 13 ,
static_cast<int>( KEYWORDS::SYMBOL) , 4  );
    /// 'table'
    mark_cells( table , 4  ,
static_cast<int>( KEYWORDS::SYMBOL) , 15 );
    /// values
    mark_cells( table , 15 ,
static_cast<int>( KEYWORDS::SYMBOL) , 11  );
    /// "Tom"
    mark_cells( table , 11  ,
static_cast<int>( KEYWORDS::SYMBOL) , 15 );
    /// for commas
    set_success( table, 11 );
}


//===========================================================
// display functions
//===========================================================

void print_cmd( const int table[][ CMD_COL ], int row, int col)
{
    cout << left  << setfill(' ')  << setw( 5 ) << ' ' ;
    for ( int i = 0; i < col ; i++)
    {
//        if ( i == 0 ||  ( i > 40 && i <= 70 ) )
            cout << setw ( 4 ) << setfill(' ') << i   ;
    }
    cout << "\n\n";

    for ( int i = 0; i < row/2 ; i++)
    {
        /// row numbers printed on the side
        cout << left  << setfill(' ')  << setw( 5 ) << i ;

        for ( int j = 0; j < col; j++ )
        {
//            if ( j == 0 || ( j > 40  && j <= 70 ) )
            {
                cout << setfill(' ')  << setw( 4 )  << table[i]
[j] ;
            }
        }
        cout << "\n\n";
    }
}

void show_string(char s[], unsigned int pos )
{
    // print s to the screen and point to the character at pos
with ^
    for ( size_t i = 0; i < strlen(s); i++ )
    {
        cout << s[i];
    }
    cout << endl << setw( pos ) << "^" << "\t pos = " << pos <<
endl;

}
```