```cpp
/*
 *  LanguageDescriptor.h
 *  Defines the Language Descriptor class, which is the bridge between a text language descriptor file
 *
 *
 *  Created: 1/3/2017 by Ryan Tedeschi
 */

#ifndef LANGUAGE_DESCRIPTOR_H
#define LANGUAGE_DESCRIPTOR_H

#include <vector>
#include <string>
#include <sstream>
#include <regex>
#include <iostream>
#include <unordered_map>
#include "../Markup/Markup.h"
#include "../Helpers/Helpers.h"

#define CFG_EXT ".cfg"
#define CFG_DIR "./cfg/"

using namespace std;

enum ProductionSetType { _Root, _Terminal, _Group, _Alternation, _Production, _Action };

class Production;
class ProductionSet;
class LanguageDescriptorObject;
class TokenMatch;

class ActionRoutine {
    public:
        virtual Markup* Execute(Markup*, vector<Markup*>) = 0;
};

class DeclareVarAction : public ActionRoutine {
    public:
        Markup* Execute(Markup*, vector<Markup*>);
};
class AssignVarAction : public ActionRoutine {
    public:
        Markup* Execute(Markup*, vector<Markup*>);
};
class AccumulateVarAction : public ActionRoutine {
    public:
        Markup* Execute(Markup*, vector<Markup*>);
};
class ResolveExprAction : public ActionRoutine {
    public:
        Markup* Execute(Markup*, vector<Markup*>);

    private:
        Markup* ResolveExpr(Markup*);
};

class ActionRoutines {
    public:
        static Markup* ExecuteAction(string, Markup*);
        static Markup* ExecuteAction(string, Markup*, vector<Markup*>);

    private:
        static vector<Markup*> ResolveParameters(string, Markup*);
        static Markup* ResolveParameter(string, Markup*);

        static unordered_map<string, ActionRoutine*> actions;
};

class Token : public Printable {
    public:
        Token(string, string);
        string id;
        string value;

        void Print();

    private:

};

class LanguageDescriptorObject
{
    public:
        LanguageDescriptorObject(string);
        LanguageDescriptorObject();
        ~LanguageDescriptorObject();

        vector<Token> Tokenize(string);
        vector<Token> Tokenize(Markup*);

        void Parse(string);

        string LookupTerminalValue(string);
        bool IsTerminalIgnored(string);
        Production* findProdById(string);
        int getProdIndex(string);
        vector<Production*> GetOrderedProductions(vector<string>);
        vector<Production*> GetProductions();

        string GetLanguage();


    private:
        void ParseTerminalValues(string);
        void ParseFSM(string);
        void ParseReservedWords(string);
        void ParseIgnores(string);

        unordered_map<string, bool> ignore;
        unordered_map<string, string> terminals;
        unordered_map<string, string> reservedWords;
        vector<Production*> productions;
        FSM<char> stateMachine;
        string language;
};

class TokenMatch {
    public:
        bool isAction = false;
        string prod;
        int begin;
        int end;
        int length;
        vector<Token> match;
        vector<TokenMatch*> submatches;

        Markup* GenerateMarkup(Markup* parent = NULL, bool addChildrenToParent = false);
        void Print(int);

    private:

};
```

```

class ProductionSet {
    public:
        ProductionSet(Production*);
        void Parse(string);
        void SetAction(string);
        void SetTerminal(string);
        void SetProduction(string);
        void SetAlternation(string);
        void SetMultiplicity(string);

        TokenMatch* MatchStrict(vector<Token>, int);
        TokenMatch* Match(vector<Token>, int);
        TokenMatch* Match(vector<Token>);

        Production* GetProduction();

        ProductionSetType GetType();
        vector<ProductionSet*> GetChildren();
        string GetSource();
        string GetMultiplicity();

        // Markup Parser(vector<string>);

    private:
        TokenMatch* MatchGroup(vector<Token>, int);
        TokenMatch* MatchTerminal(vector<Token>, int);
        TokenMatch* MatchAlternation(vector<Token>, int);
        TokenMatch* MatchProduction(vector<Token>, int);
        TokenMatch* MatchAction(string, int);

        Production* prod;
        enum ProductionSetType type = _Root;
        string source = "";
        vector<ProductionSet*> children;
        string multiplicity = "";
};

class Production {
    public:
        Production(LanguageDescriptorObject*, string, string);
        void Parse(string, string);

        LanguageDescriptorObject* GetLDO();
        ProductionSet* GetRootProductionSet();
        TokenMatch* Match(vector<Token>);
        TokenMatch* Match(vector<Token>, int);
        TokenMatch* MatchStrict(vector<Token>);
        TokenMatch* MatchStrict(vector<Token>, int);

        string GetRegex();
        string GetId();
        vector<Production*> GetContainedProductions();

    private:
        LanguageDescriptorObject* ldo;
        string id;
        string data;
        vector<string> subproductions;
        ProductionSet* rootSet;
};

#endif
```