EXP 9
13/04/21

COMPUTATION OF LR(0) ITEMS

AIM: To implement LR(0) Parser in C++.

ALGORITHM:

1. Start.
2. Create structure for production with LHS and RHS.
3. Open file and read input from file.
4. Build state 0 from extra grammar Law S' -> S $ that is all start symbol of grammar and one Dot (.) before S symbol.
5. If Dot symbol is before a non-terminal, add grammar laws that this non-terminal is in Left Hand Side of that Law and set Dot in before of first part of Right Hand Side.
6. If state exists (a state with this Laws and same Dot position), use that instead.
7. Now find set of terminals and non-terminals in which Dot exist in before.
8. If step 7 Set is non-empty go to 9, else go to 10.
9. For each terminal/non-terminal in set step 7 create new state by using all grammar law that Dot position is before of that terminal/non-terminal in reference state by increasing Dot point to next part in Right Hand Side of that laws.
10. Go to step 5.
11. End of state building.
12. Display the output.
13. End.

CODE:

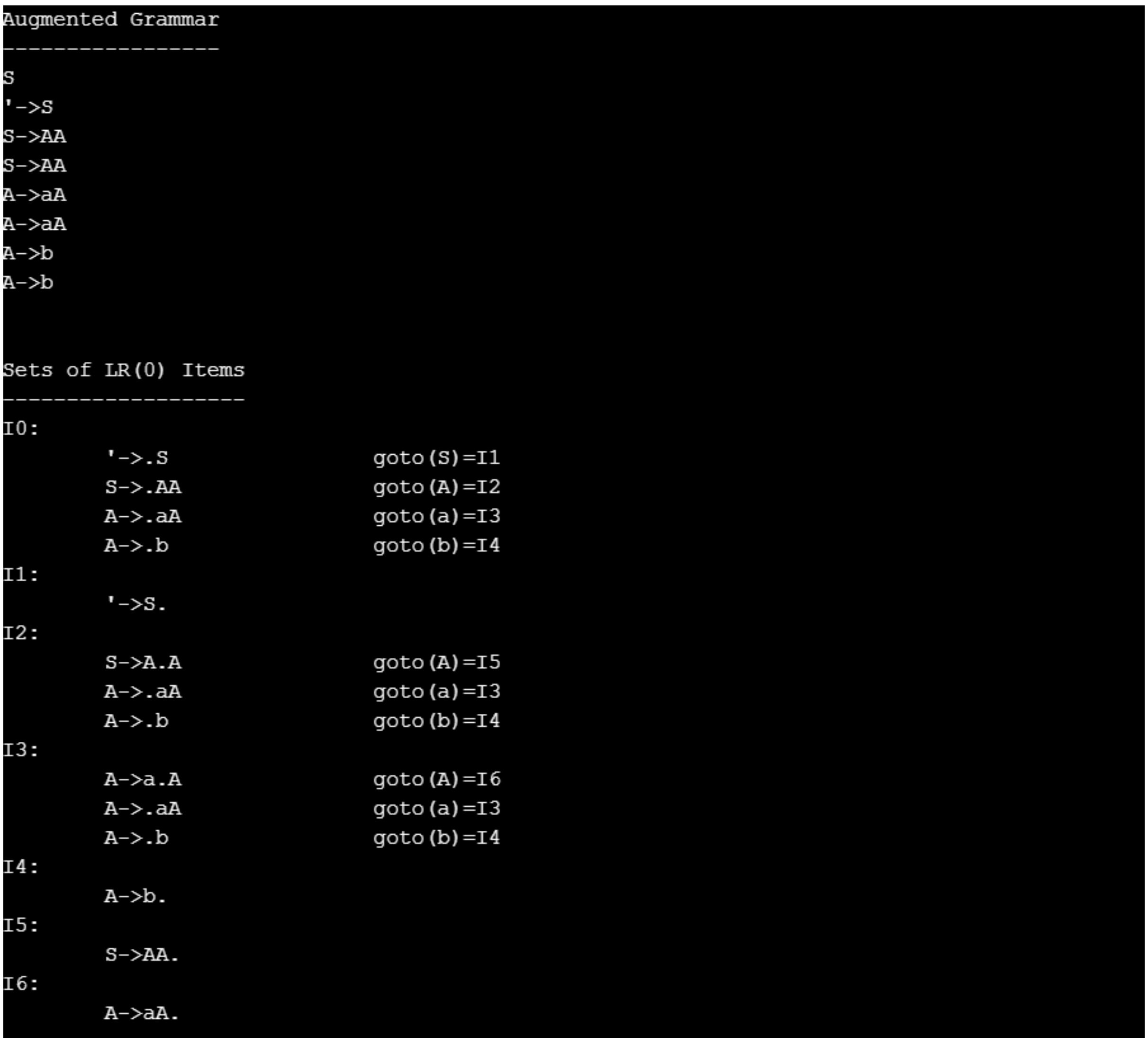
```
#include <cstdlib>
#include <iostream>
#include <string>
#include <string.h>
#include <vector>
#include <algorithm>
#include <map>

using namespace std;

typedef map<char, vector<string> > AugmentedGrammar;
typedef map<string, int> GotoMap;
struct AugmentedProduction
{
   char lhs;
   string rhs;

   AugmentedProduction() {}
```

```
    AugmentedProduction(char _lhs, string _rhs) : lhs(_lhs), rhs(_rhs) {}
};

class LR0Item
{
private:
    vector<AugmentedProduction*> productions;

public:
    map<char, int> gotos;

    LR0Item() {}
    ~LR0Item() {}

    void Push(AugmentedProduction *p) { productions.push_back(p); }
    int Size() { return int(productions.size()); }

    bool Contains (string production) {
        for (auto it = productions.begin(); it != productions.end(); it++) {
            string existing = string(&(*it)->lhs, 1) + "->" + (*it)->rhs;
            if (strcmp(production.c_str(), existing.c_str()) == 0) {
                return true;
            }
        }
        return false;
    }

    AugmentedProduction* operator[](const int index) {
        return productions[index];
    }
};

void
add_closure(char lookahead, LR0Item& item, AugmentedGrammar& grammar)
{
    if (!isupper(lookahead)) return;

    string lhs = string(&lookahead, 1);
    for (int i = 0; i<grammar[lookahead].size(); i++) {
        string rhs = "." + grammar[lookahead][i];
        if (!item.Contains( lhs + "->" + rhs )) {
            item.Push(new AugmentedProduction(lookahead, rhs));
        }
    }
```

```
}

void
get_LR0_items(vector<LR0Item>& lr0items, AugmentedGrammar& grammar, int&
itemid, GotoMap& globalGoto)
{
   printf("I%d:\n", itemid);

   for (int i = 0; i<lr0items[itemid].Size(); i++) {
      string rhs = lr0items[itemid][i]->rhs;
      char lookahead = rhs[rhs.find('.')+1];
      add_closure(lookahead, lr0items[itemid], grammar);
   }

   int nextpos;
   char lookahead, lhs;
   string rhs;
   AugmentedProduction *prod;

   for (int i = 0; i<lr0items[itemid].Size(); i++) {
      lhs = lr0items[itemid][i]->lhs;
      rhs = lr0items[itemid][i]->rhs;
      string production = string(&lhs,1) + "->" + rhs;

      lookahead = rhs[rhs.find('.')+1];
      if (lookahead == '\0') {
         printf("\t%-20s\n", &production[0]);
         continue;
      }

      if (lr0items[itemid].gotos.find(lookahead) == lr0items[itemid].gotos.end()) {
         if (globalGoto.find(production) == globalGoto.end()) {
            lr0items.push_back(LR0Item()); // create new state (item)
            string newRhs = rhs;
            int atpos = newRhs.find('.');
            swap(newRhs[atpos], newRhs[atpos+1]);
            lr0items.back().Push(new AugmentedProduction(lhs, newRhs));
            lr0items[itemid].gotos[lookahead] = lr0items.size()-1;
            globalGoto[production] = lr0items.size()-1;
         } else {
            lr0items[itemid].gotos[lookahead] = globalGoto[production];
         }
         printf("\t%-20s goto(%c)=I%d\n", &production[0], lookahead,
globalGoto[production]);
```

```
        } else {
            int at = rhs.find('.');
            swap(rhs[at], rhs[at+1]);
            int nextItem = lr0items[itemid].gotos[lookahead];
            if (!lr0items[nextItem].Contains(string(&lhs, 1) + "->" + rhs)) {
                lr0items[nextItem].Push(new AugmentedProduction(lhs, rhs));
            }
            swap(rhs[at], rhs[at+1]);
            printf("\t%-20s\n", &production[0]);
        }
    }
}

/**
 * void load_grammar
 * scan and load the grammar from stdin while setting first LR(0) item */
void load_grammar(AugmentedGrammar& grammar, vector<LR0Item>& lr0items)
{
    string production;
    string lhs, rhs;
    string delim = "->";

    getline(cin, lhs); // scan start production
    grammar['\''].push_back(lhs);
    lr0items[0].Push(new AugmentedProduction('\'', "." + lhs));
    printf("'->%s\n", lhs.c_str());

    while(1) {
        getline(cin, production);
        if (production.length() < 1) return;

        auto pos = production.find(delim);
        if(pos!=string::npos){
            lhs = production.substr(0,pos);
            rhs = production.substr(pos+delim.length(),std::string::npos);
        }

        grammar[lhs[0]].push_back(rhs);
        printf("%s->%s\n", lhs.c_str(), rhs.c_str());
        lr0items[0].Push(new AugmentedProduction(lhs[0], "." + rhs));
    }
}

int main() {
    int itemid = -1; // counter for the number of LR(0) items
    AugmentedGrammar grammar;
    vector<LR0Item> lr0items = { LR0Item() }; // push start state
    GotoMap globalGoto;

    printf("Augmented Grammar\n");
    printf("-----------------\n");
    load_grammar(grammar, lr0items);
    printf("\n");

    printf("Sets of LR(0) Items\n");
    printf("-------------------\n");
    while (++itemid < int(lr0items.size())) {
        get_LR0_items(lr0items, grammar, itemid, globalGoto);
    }
    printf("\n");
    return 0;
}
```

OUTPUT:



```
Augmented Grammar
-----------------
S
'->S
S->AA
S->AA
A->aA
A->aA
A->b
A->b


Sets of LR(0) Items
-------------------
I0:
        '->.S                   goto(S)=I1
        S->.AA                  goto(A)=I2
        A->.aA                  goto(a)=I3
        A->.b                   goto(b)=I4
I1:
        '->S.
I2:
        S->A.A                  goto(A)=I5
        A->.aA                  goto(a)=I3
        A->.b                   goto(b)=I4
I3:
        A->a.A                  goto(A)=I6
        A->.aA                  goto(a)=I3
        A->.b                   goto(b)=I4
I4:
        A->b.
I5:
        S->AA.
I6:
        A->aA.
```

RESULT: Implementation of LR(0) Parser was compiled, executed and verified successfully.