THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

LAB - ASSIGNMENT- 2

COMPILER CONSTRUCTION

UCS802

SUBMITTED BY-	**SUBMITTED TO-**
GOURISH SINGLA	**Dr. RAJ KUMAR TEKCHANDANI**
101803698	
COE-15	

Code:

```
#include<bits/stdc++.h>
using namespace std;

class production
{
	public:
	char left;
	string right;
};
char start_nt;
vector<production> get_productions(vector<string> input)
{
	vector<production> pros;
	for(int i=0;i<input.size();i++)
	{
		production p;
		p.left=input[i][0];
		vector<string> rights;
		string s="";
		int flag=0;
		for(int j=1;j<input[i].length();j++)
		{
			if(input[i][j]=='-' and input[i][j+1]=='>')
			{
				j+=1;
				continue;
			}
			if(input[i][j]=='/')
			{
				flag=1;
				rights.push_back(s);
				s.clear();
				continue;
			}
			s+=input[i][j];
		}

		rights.push_back(s);
		for(int j=0;j<rights.size();j++)
		{
			p.right=rights[j];
			pros.push_back(p);
		}
	}
	return pros;
}
vector<char> get_terminals(vector<production> pros)
{
	vector<char> t;
	for(int i=0;i<pros.size();i++)
	{
		for(int j=0;j<pros[i].right.length();j++)
		{
			if(find(t.begin(),t.end(),pros[i].right[j])==t.end())
				if(pros[i].right[j]<65 or pros[i].right[j]>90)
					t.push_back(pros[i].right[j]);
		}
```

```
    }
    return t;
}
vector<char> non_terminal;
vector<char> get_non_terminals(vector<production> pros)
{
    vector<char> nt;
    for(int i=0;i<pros.size();i++)
    {
        for(int j=0;j<pros[i].right.length();j++)
        {
            if(find(nt.begin(),nt.end(),pros[i].right[j])==nt.end())
                if(pros[i].right[j]>=65 and pros[i].right[j]<90)
                    nt.push_back(pros[i].right[j]);
        }
    }
    return nt;
}

class fnfs
{
    public:
        char left;
        vector<char> fs;
};

vector<fnfs> get_firsts(vector<char> non_terminal,vector<char> terminal,vector<production> pros)
{
    vector<fnfs> res;
    for(int i=0;i<non_terminal.size();i++)
    {
        fnfs a;
        a.left=non_terminal[i];
        vector<char> s,nt;
        nt.push_back(non_terminal[i]);
        while(nt.size()>0)
        {
            char z=nt.back();
            nt.pop_back();
            for(int j=0;j<pros.size();j++)
            {
                if(pros[j].left==z)
                {
                    if(pros[j].right[0]!=z)
                    {
                        if(find(terminal.begin(),terminal.end(),pros[j].right[0])==terminal.end())
                        {
                            nt.push_back(pros[j].right[0]);
                        }
                        else if(find(s.begin(),s.end(),pros[j].right[0])==s.end())
                        {
                            s.push_back(pros[j].right[0]);
                        }
                    }
                }
                else
                continue;
            }
        }
        a.fs=s;
        res.push_back(a);
    }
    return res;
}
vector<fnfs> follows;
vector<fnfs> get_follows(vector<char> non_terminal,vector<char> terminal,vector<fnfs> firsts,vector<production> pros)
{
    vector<fnfs> res;

    for(int l=0;l<non_terminal.size();l++)
    {
        fnfs a;
        a.left=non_terminal[l];
        vector<char> s;
        if(a.left==start_nt)
        s.push_back('$');

        for(int i=0;i<pros.size();i++)
        {
            int found=0;
            for(int j=0;j<pros[i].right.size();j++)
            {
                if(pros[i].right[j]==a.left)
                {
                    found=1;
                }
                if(found==1)
                {
                    if(j+1==pros[i].right.size())
                    {
                        for(int k=0;k<res.size();k++)
                        {
                            if(res[k].left==pros[i].left)
                            {
                                for(int h=0;h<res[k].fs.size();h++)
                                {
                                    if(find(s.begin(),s.end(),res[k].fs[h])==s.end())
                                        s.push_back(res[k].fs[h]);
                                }
                            }
                        }
                    }
                    else if(find(terminal.begin(),terminal.end(),pros[i].right[j+1])!=terminal.end())
                    {
                        s.push_back(pros[i].right[j+1]);
                    }
                    else if(find(non_terminal.begin(),non_terminal.end(),pros[i].right[j+1])!=non_terminal.end())
                    {
                        for(int y=0;y<firsts.size();y++)
                        {
                            if(firsts[y].left==pros[i].right[j+1])
                            {
                                for(int x=0;x<firsts[y].fs.size();x++)
                                {
                                    if(find(s.begin(),s.end(),firsts[y].fs[x])==s.end())
                                    s.push_back(firsts[y].fs[x]);
                                }
                            }
                        }
                    }
                }
                found=0;
            }
        }
        a.fs=s;
        res.push_back(a);
    }
    return res;
}

vector<production> get_pros_ready(vector<production> pros)
{
    for(int i=0;i<pros.size();i++)
    {
        pros[i].right='.'+pros[i].right;
    }
    return pros;
}

vector<production> refer;

int num=0;
class node
{
    public:
        vector<production> my_state,next_state;
        char read;

};

vector<production> get_from_refer(char nt)
{
    vector<production> res;
    vector<char> stk;
    stk.push_back(nt);
    while(stk.size()>0)
    {
        char z=stk.back();
        stk.pop_back();
        for(int i=0;i<refer.size();i++)
        {
            if(refer[i].left==z)
            {
                res.push_back(refer[i]);
                for(int j=0;j<refer[i].right.length();j++)
                {
                    if(refer[i].right[j]=='.')
                    {
                        if(refer[i].right[j+1]==z)
                        continue;

    if(find(non_terminal.begin(),non_terminal.end(),refer[i].right[j+1])!=non_terminal.end())
                        {
                            stk.push_back(refer[i].right[j+1]);
                        }
                    }
                }
            }
        }
    }
    return res;
}

bool present(vector<production> test,production a)
{
    for(int i=0;i<test.size();i++)
    {
        if(test[i].left==a.left and test[i].right==a.right)
        return true;
    }
    return false;
}

vector<production> dot_shifter(vector<production> test)
{
    for(int i=0;i<test.size();i++)
    {
        for(int j=0;j<test[i].right.length();j++)
        {
            if(test[i].right[j]=='.' and j!=test[i].right.length()-1)
            {
                char temp=test[i].right[j];
                test[i].right[j]=test[i].right[j+1];
                test[i].right[j+1]=temp;
                break;
            }
        }
    }

    for(int i=0;i<test.size();i++)
    {
        for(int j=0;j<test[i].right.length();j++)
        {
            if(test[i].right[j]=='.' and j!=test[i].right.length()-1)
            {
                if(find(non_terminal.begin(),non_terminal.end(),test[i].right[j+1])!=non_terminal.end())
                {
                    vector<production> a=get_from_refer(test[i].right[j+1]);
                    for(int k=0;k<a.size();k++)
                    {
                        if(present(test,a[k])==false)
                        {
                            test.push_back(a[k]);
                        }
                    }
                }
            }
        }
    }
    return test;
}
```

```
bool already_found(vector<production> p,vector<node> graph)
{
    for(int i=0;i<graph.size();i++)
    {
        int found=0;
        if(graph[i].my_state.size()==p.size())
        {
            for(int j=0;j<graph[i].my_state.size();j++)
            {
                if(p[j].left==graph[i].my_state[j].left and p[j].right==graph[i].my_state[j].right)
                {
                    found++;
                }
            }
            if(found==graph[i].my_state.size())
            return true;
        }
    }
    return false;
}

vector<node> get_graph(vector<production> test)
{
    vector<node> res;
    vector<vector<production> > queue;
    queue.push_back(test);
    while(queue.size()>0)
    {
        vector<production> pros=queue[0],test=queue[0];
        queue.erase(queue.begin());
        for(int i=0;i<pros.size();i+=1)
        {
            node i0;
            i0.my_state=test;
            if(pros[i].right[pros[i].right.length()-1]=='.')
            {
                pros.erase(pros.begin()+i);
                i--;
                continue;
            }
            vector<production> to_pick;
            to_pick.push_back(pros[i]);
            char ch;
            for(int j=0;j<to_pick.back().right.length()-1;j++)
            {
                if(to_pick.back().right[j]=='.')
                {
                    ch=to_pick.back().right[j+1];
                    if(find(non_terminal.begin(),non_terminal.end(),ch)!=non_terminal.end())
                    {
                        for(int temp=i+1;temp<pros.size();temp+=1)
                        {
                            for(int k=0;k<pros[temp].right.length()-1;k++)
                            {
                                if(pros[temp].right[k]=='.' and pros[temp].right[k+1]==ch)
                                {
                                    to_pick.push_back(pros[temp]);
                                    pros.erase(pros.begin()+temp);
```

```
									temp-=1;
									break;

								}
							}
						}
					}
				}
			}
			pros.erase(pros.begin()+i);
			i-=1;
			vector<production> next=dot_shifter(to_pick);
			if(already_found(next,res)==false)
			queue.push_back(next);
			i0.next_state=next;
			i0.read=ch;
			res.push_back(i0);
		}
	}
	return res;
}

class naming
{
	public:
		vector<production> state;
		int val;
		production complete;
		int len_to_reduce;
		naming()
		{
			complete.right='?';
			complete.left='?';
			len_to_reduce=0;
		}
};

bool same_state(vector<production> test1,vector<production> test2)
{
	if(test1.size()!=test2.size())
	return false;
	int found=0;
	for(int i=0;i<test1.size();i++)
	{
		if(test1[i].left==test2[i].left and test1[i].right==test2[i].right)
		found++;
		else
		return false;
	}
	if(found==test1.size())
	return true;
	return false;
}
```

```
vector<naming> give_names(vector<node> graph)
{
    vector<naming> res;
    for(int i=0;i<graph.size();i++)
    {
        node test1=graph[i];
        int found=0;
        for(int j=0;j<res.size();j++)
        {
            naming test2=res[j];
            if(same_state(test1.my_state,test2.state))
            {
                found=1;
                break;
            }
        }
        if(found==0)
        {
            naming t;
            t.state=test1.my_state;
            t.val=num++;
            res.push_back(t);
        }
        found=0;
        for(int j=0;j<res.size();j++)
        {
            naming test2=res[j];
            if(same_state(test1.next_state,test2.state))
            {
                found=1;
                break;
            }
        }
        if(found==0)
        {
            naming t;
            t.state=test1.next_state;
            t.val=num++;
            res.push_back(t);
        }
    }
    for(int i=0;i<res.size();i++)
    {
        for(int j=0;j<res[i].state.size();j++)
        {
            if(res[i].state[j].right[res[i].state[j].right.length()-1]=='.')
            {
                res[i].complete=res[i].state[j];
                res[i].len_to_reduce=res[i].state[j].right.length()-1;
                break;
            }
        }
    }
    return res;
}
```

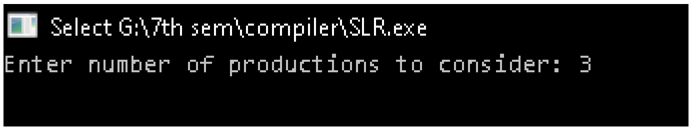
```
int get_state_val(vector<production> pro,vector<naming> states)
{
    for(int i=0;i<states.size();i++)
    {
        naming test=states[i];
        if(same_state(pro,test.state))
        return test.val;
    }
    return -1;
}

class compact_node
{
    public:
        int start,end;
        char read;
};

int get_next(int st_top,char r,vector<compact_node> fg)
{
    for(int i=0;i<fg.size();i++)
    {
        if(fg[i].start==st_top and fg[i].read==r)
        {
            return fg[i].end;
        }
    }
    return -1;
}
vector<production> final_parsing;
string parser(vector<naming> all_states,vector<compact_node> fg,string test)
{
    final_parsing;

    vector<char> stack;
    test=test+"$";
    vector<char> input;
    stack.push_back((char)0);
    for(int i=0;i<test.length();i++)
    {
        input.push_back(test[i]);
    }
    int next;
    while(stack.size()>0)
    {
        if(stack.back()==1 and input[0]=='$')
        {
            return "Accept";
        }

        if(find(non_terminal.begin(),non_terminal.end(),stack[stack.size()-1])!=non_terminal.end())
        {
            for(int i=0;i<fg.size();i++)
            {
                if(fg[i].start==stack[stack.size()-2] and fg[i].read==stack[stack.size()-1])
                {
                    stack.push_back(fg[i].end);
                    break;
                }
            }
            continue;
        }

        next=get_next(int(stack.back()),input[0],fg);
        production a;
        int len;
        if(next==-1)
        {
            int found=0;
            for(int i=0;i<all_states.size();i++)
            {
                if(all_states[i].val==stack.back())
                {
                    for(int k=0;k<follows.size();k++)
                    {
                        if(all_states[i].complete.left==follows[k].left)
                        {
                            if(find(follows[k].fs.begin(),follows[k].fs.end(),input[0])==follows[k].fs.end())
                            {
                                return "Reject";
                            }
                            else
                            {
                                found=1;
                                break;
                            }
                        }
                    }
                    len=all_states[i].len_to_reduce;
                    a=all_states[i].complete;
                    final_parsing.push_back(a);
                    for(int j=0;j<2*len;j++)
                    {
                        stack.pop_back();
                    }
                    stack.push_back(all_states[i].complete.left);
                    break;
                }
            }
            if(found==0)
                return "Reject";
        }
        else
        {
            stack.push_back(input[0]);
            stack.push_back(next);
            input.erase(input.begin());
        }
    }
    return "Reject";
}
```

```
int main()
{
        int n;
        cout<<"Enter number of productions to consider: ";
        cin>>n;
        vector<string> input;
        cout<<"Enter Productions: "<<endl;
        for(int i=0;i<n;i++)
        {
                string s;
                cin>>s;
                input.push_back(s);
        }

        for(int i=0;i<60;i++)
        cout<<"-";
        cout<<endl;
        vector<production> pros=get_productions(input);
        production a;
        a.left='Z';
        a.right=pros[0].left;
        start_nt=pros[0].left;
        pros.insert(pros.begin(),a);
        cout<<"Atomic Productions: "<<endl;
        for(int i=0;i<pros.size();i++)
        {
                cout<<pros[i].left<<" => "<<pros[i].right<<endl;
        }

        for(int i=0;i<60;i++)
        cout<<"-";
        cout<<endl;
        non_terminal=get_non_terminals(pros);
        cout<<"Non-terminals provided: ";
        for(int i=0;i<non_terminal.size();i++)
        {
                cout<<non_terminal[i]<<" ";
        }
        cout<<endl;

        for(int i=0;i<60;i++)
        cout<<"-";
        cout<<endl;
        vector<char> terminal=get_terminals(pros);
        cout<<"Terminals provided: ";
        for(int i=0;i<terminal.size();i++)
        {
                cout<<terminal[i]<<" ";
        }
        cout<<endl;

        for(int i=0;i<60;i++)
        cout<<"-";
        cout<<endl;
        vector<fnfs> firsts=get_firsts(non_terminal,terminal,pros);
        cout<<"Firsts of Non-terminals: "<<endl;
```

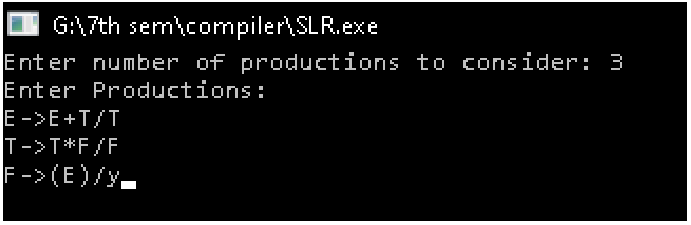
```
    for(int i=0;i<firsts.size();i++)
    {
        cout<<firsts[i].left<<" => ";
        for(int j=0;j<firsts[i].fs.size();j++)
        {
            cout<<firsts[i].fs[j]<<" ";
        }
        cout<<endl;
    }

    for(int i=0;i<60;i++)
    cout<<"-";
    cout<<endl;
    follows=get_follows(non_terminal,terminal,firsts,pros);
    cout<<"Follows of Non-terminals: "<<endl;
    for(int i=0;i<follows.size();i++)
    {
        cout<<follows[i].left<<" => ";
        for(int j=0;j<follows[i].fs.size();j++)
        {
            cout<<follows[i].fs[j]<<" ";
        }
        cout<<endl;
    }

    pros=get_pros_ready(pros);
    refer=pros;

    for(int i=0;i<60;i++)
    cout<<"-";
    cout<<endl;
    vector<node> graph=get_graph(pros);
    vector<naming> states=give_names(graph);
    cout<<"States: "<<endl;
    for(int i=0;i<states.size();i++)
    {
        cout<<"State: "<<states[i].val<<endl;
        for(int j=0;j<states[i].state.size();j++)
        {
            cout<<states[i].state[j].left<<" => "<<states[i].state[j].right<<endl;
        }
        if(states[i].complete.left!='?')
        {
            cout<<"If Complete?: Left Non_terminal: "<<states[i].complete.left<<" => "<<states[i].complete.right<<endl;
            cout<<"Length to be reduced: "<<states[i].len_to_reduce<<endl;
        }
        cout<<endl;
    }

    for(int i=0;i<60;i++)
    cout<<"-";
    cout<<endl;
    cout<<"Graph: "<<endl;
    vector<compact_node> final_graph;
    for(int i=0;i<graph.size();i++)
    {
        compact_node a;
        node test=graph[i];
        a.start=get_state_val(test.my_state,states);
        a.end=get_state_val(test.next_state,states);
        a.read=test.read;
        final_graph.push_back(a);
    }
    for(int i=0;i<final_graph.size();i++)
    {
        compact_node a=final_graph[i];
        cout<<a.start<<" => "<<a.read<<" => "<<a.end<<"  "<<endl;
    }

    for(int i=0;i<60;i++)
    cout<<"-";
    cout<<endl;
    string test;
    cout<<"Enter String to be tested: ";
    cin>>test;
    string result=parser(states,final_graph,test);
    cout<<"Result: "<<result<<endl;
    if(result=="Accept")
    {
        cout<<endl<<"Parsing Tree: "<<endl;
        for(int i=final_parsing.size()-1;i>=0;i--)
        {
            final_parsing[i].right.erase(final_parsing[i].right.size() - 1);
            cout<<final_parsing[i].left<<" => "<<final_parsing[i].right<<endl;
        }
    }
    return 0;
}
```

Output:

INPUT

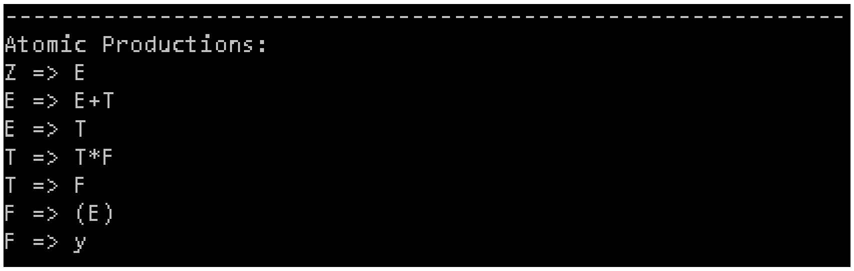


```
Enter number of productions to consider: 3
```

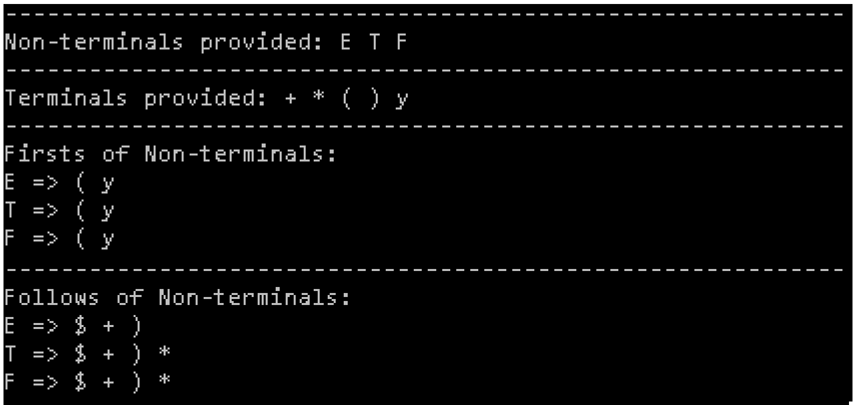


```
Enter number of productions to consider: 3
Enter Productions:
E->E+T/T
T->T*F/F
F->(E)/y
```

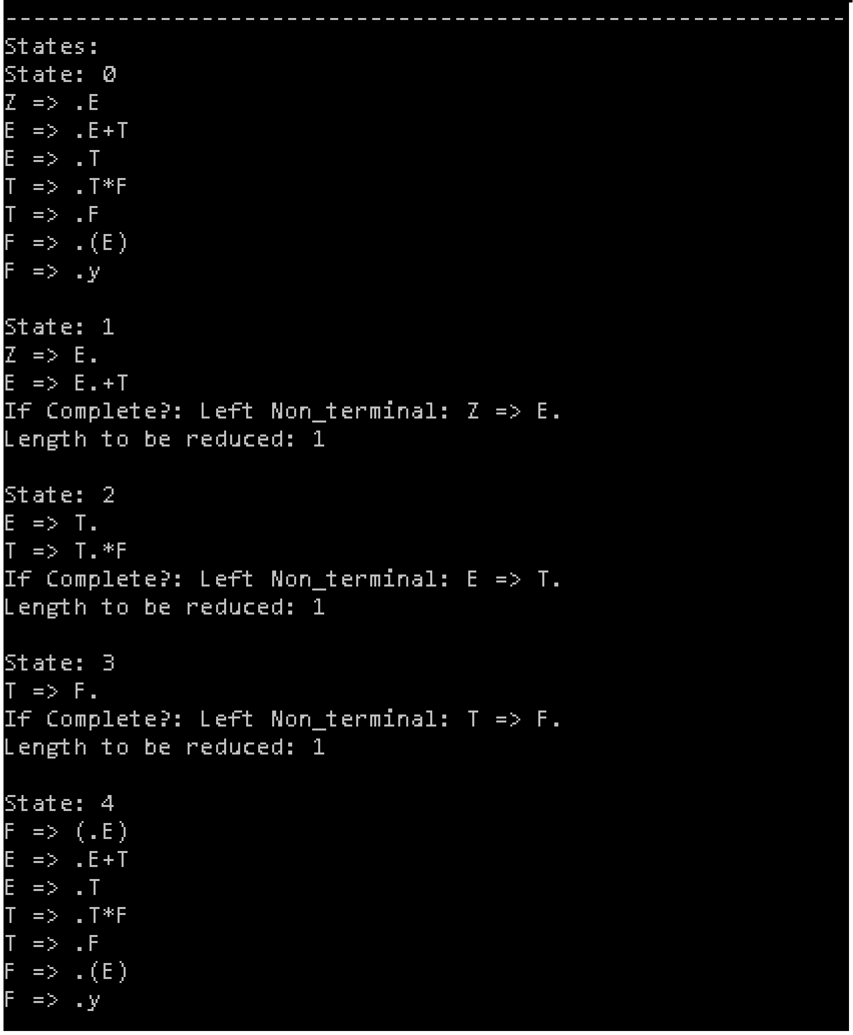
OUTPUT:



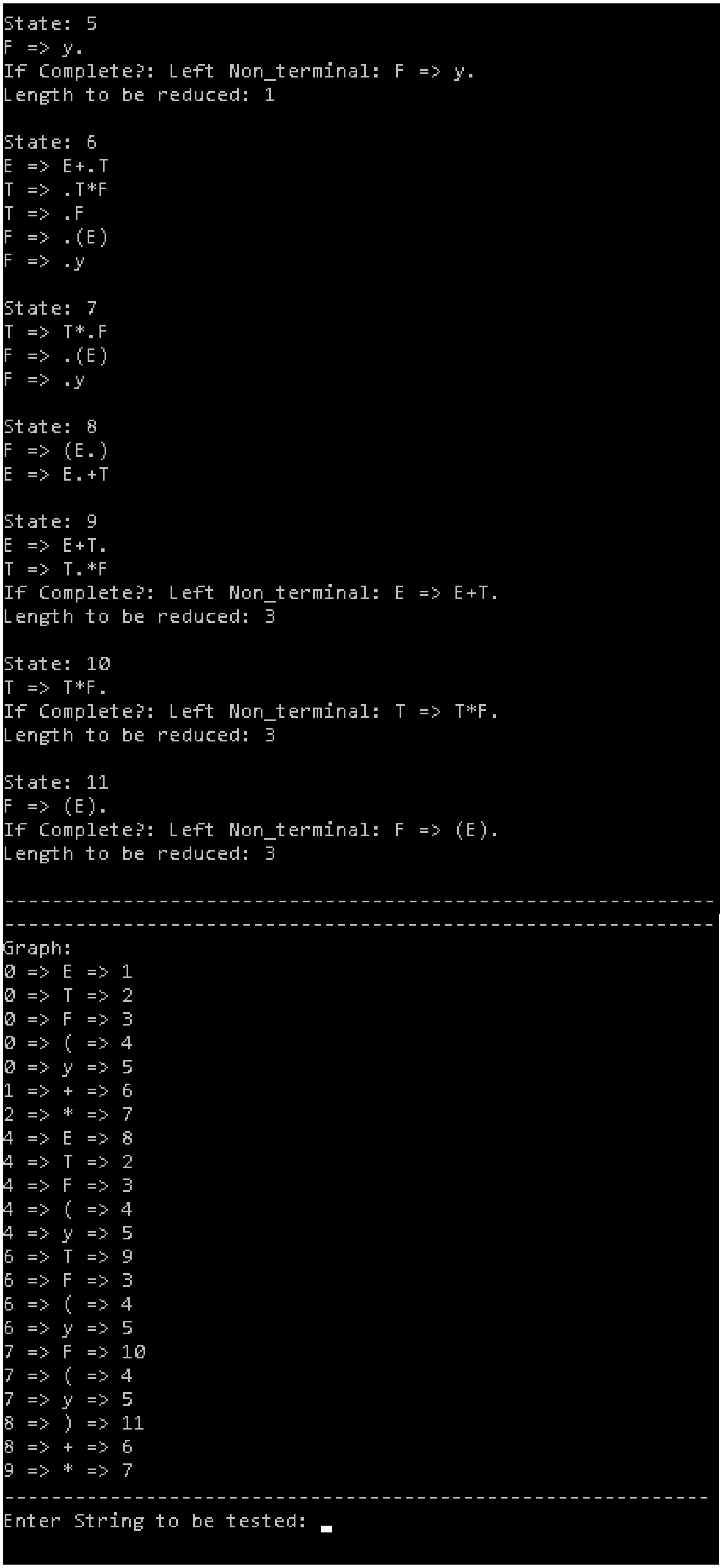
```
------------------------------------------------------------
Atomic Productions:
Z => E
E => E+T
E => T
T => T*F
T => F
F => (E)
F => y
```



```
------------------------------------------------------------
Non-terminals provided: E T F
------------------------------------------------------------
Terminals provided: + * ( ) y
------------------------------------------------------------
Firsts of Non-terminals:
E => ( y
T => ( y
F => ( y
------------------------------------------------------------
Follows of Non-terminals:
E => $ + )
T => $ + ) *
F => $ + ) *
```

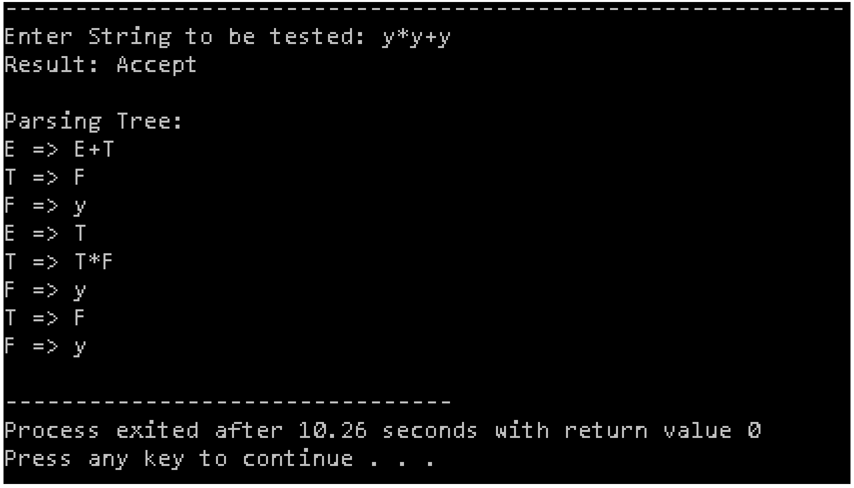


```
------------------------------------------------------------
States:
State: 0
Z => .E
E => .E+T
E => .T
T => .T*F
T => .F
F => .(E)
F => .y

State: 1
Z => E.
E => E.+T
If Complete?: Left Non_terminal: Z => E.
Length to be reduced: 1

State: 2
E => T.
T => T.*F
If Complete?: Left Non_terminal: E => T.
Length to be reduced: 1

State: 3
T => F.
If Complete?: Left Non_terminal: T => F.
Length to be reduced: 1

State: 4
F => (.E)
E => .E+T
E => .T
T => .T*F
T => .F
F => .(E)
F => .y
```



```
State: 5
F => y.
If Complete?: Left Non_terminal: F => y.
Length to be reduced: 1

State: 6
E => E+.T
T => .T*F
T => .F
F => .(E)
F => .y

State: 7
T => T*.F
F => .(E)
F => .y

State: 8
F => (E.)
E => E.+T

State: 9
E => E+T.
T => T.*F
If Complete?: Left Non_terminal: E => E+T.
Length to be reduced: 3

State: 10
T => T*F.
If Complete?: Left Non_terminal: T => T*F.
Length to be reduced: 3

State: 11
F => (E).
If Complete?: Left Non_terminal: F => (E).
Length to be reduced: 3

------------------------------------------------------------
------------------------------------------------------------
Graph:
0 => E => 1
0 => T => 2
0 => F => 3
0 => ( => 4
0 => y => 5
1 => + => 6
2 => * => 7
4 => E => 8
4 => T => 2
4 => F => 3
4 => ( => 4
4 => y => 5
6 => T => 9
6 => F => 3
6 => ( => 4
6 => y => 5
7 => F => 10
7 => ( => 4
7 => y => 5
8 => ) => 11
8 => + => 6
9 => * => 7
------------------------------------------------------------
Enter String to be tested:
```

INPUT string to be tested



```
------------------------------------------------------------
Enter String to be tested: y*y+y
Result: Accept

Parsing Tree:
E => E+T
T => F
F => y
E => T
T => T*F
F => y
T => F
F => y

--------------------------------
Process exited after 10.26 seconds with return value 0
Press any key to continue . . .
```