CSD311: Artificial Intelligence
Assignment 4

Done by:
Samarth Gupta - 1910110338
Ridham Bhagat - 1910110316
A Aadhavan - 1910110001

Documentation:

Goal-Stack Algorithm:

Given an initial state S, we are required to come up with a plan of achieving a goal state G.

Goal state G can be broken down into a series of subgoals, which together satisfy G.

$$G = G1 \wedge G2 \wedge G3 \wedge \ldots \wedge Gn.$$

Arrange these subgoals of G in the form of a stack.

```
G1
G2
G3
.
.
.
Gn
G1 ∧ G2 ∧ G3 ∧…∧ Gn
```

As it is a stack, the subgoal at the top of the stack is always worked on first.

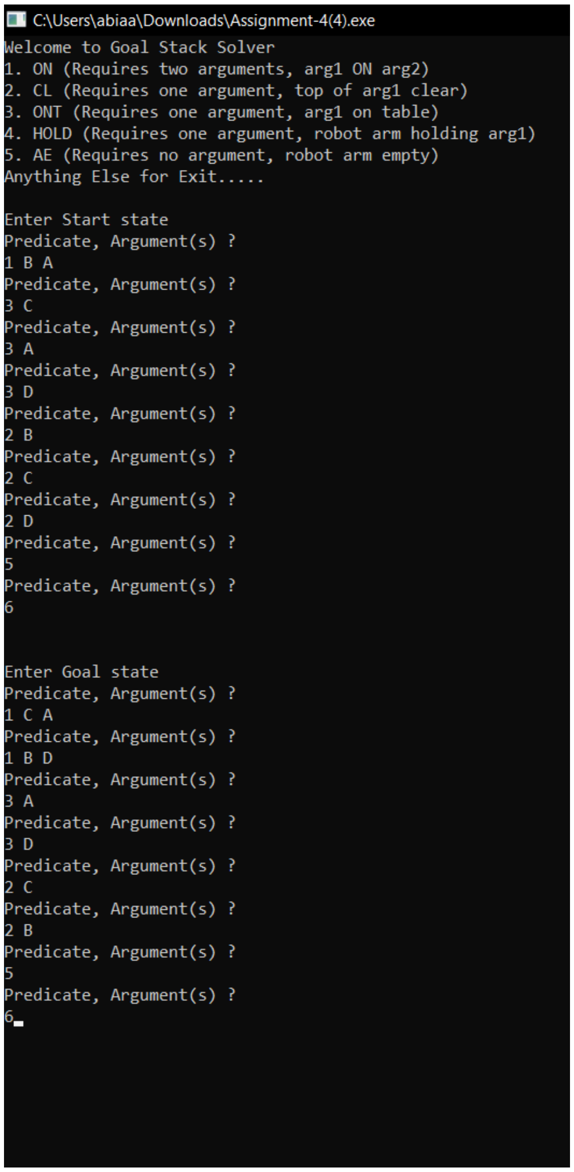
Algorithm:

```
● Push the subgoals of the goal state to the stack.
● Loop till the stack is empty:
          Pop an element E from the stack:
                If E is a predicate:
                      {
                      Push relevant action onto stack.
                      Push the individual predicates of the precondition of the action
                      onto the stack.
                      }
                Else if E is an operation:
```

```
{
Apply action on the current state and add the action to the plan.
}
```

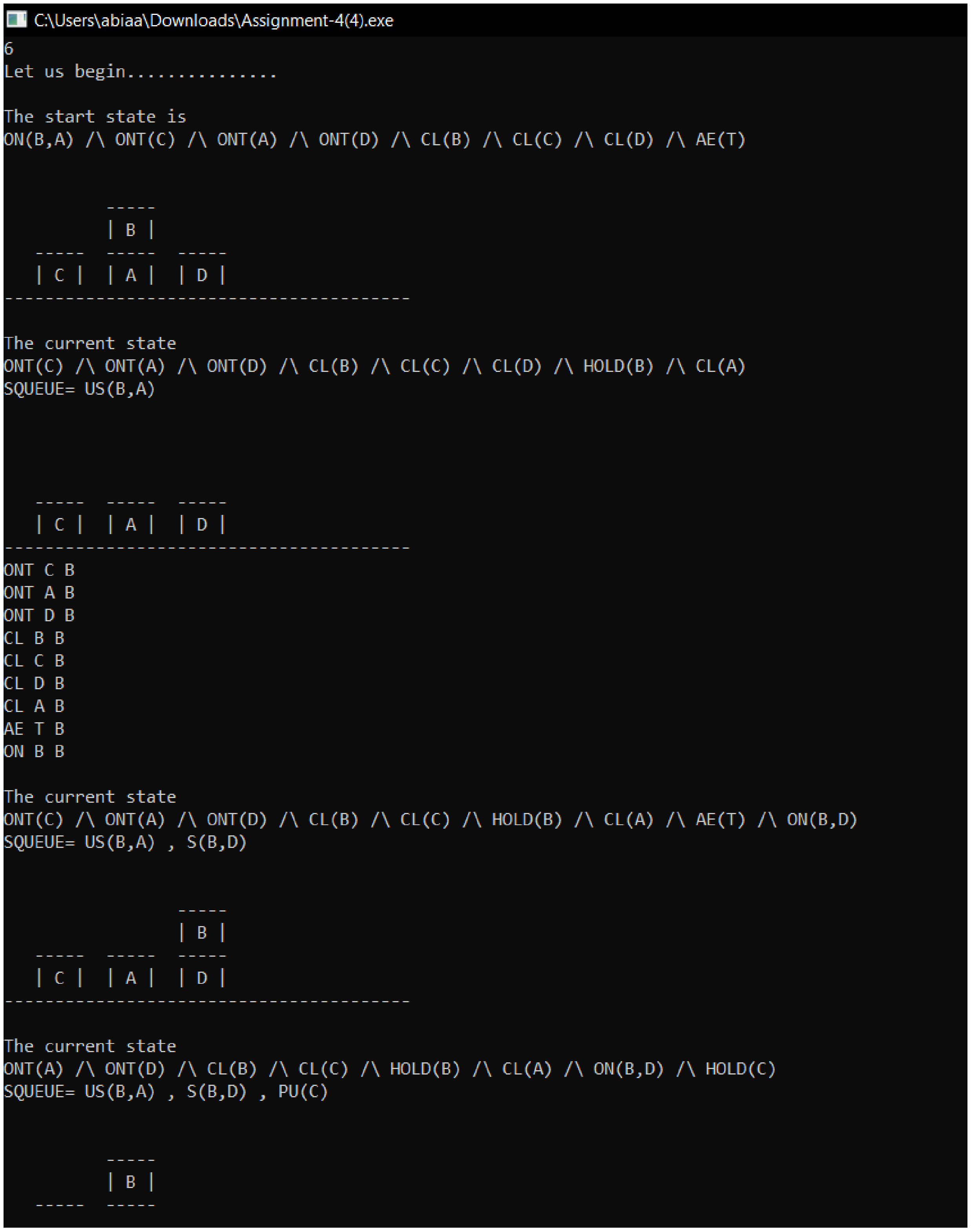
Working of Code:

Input:

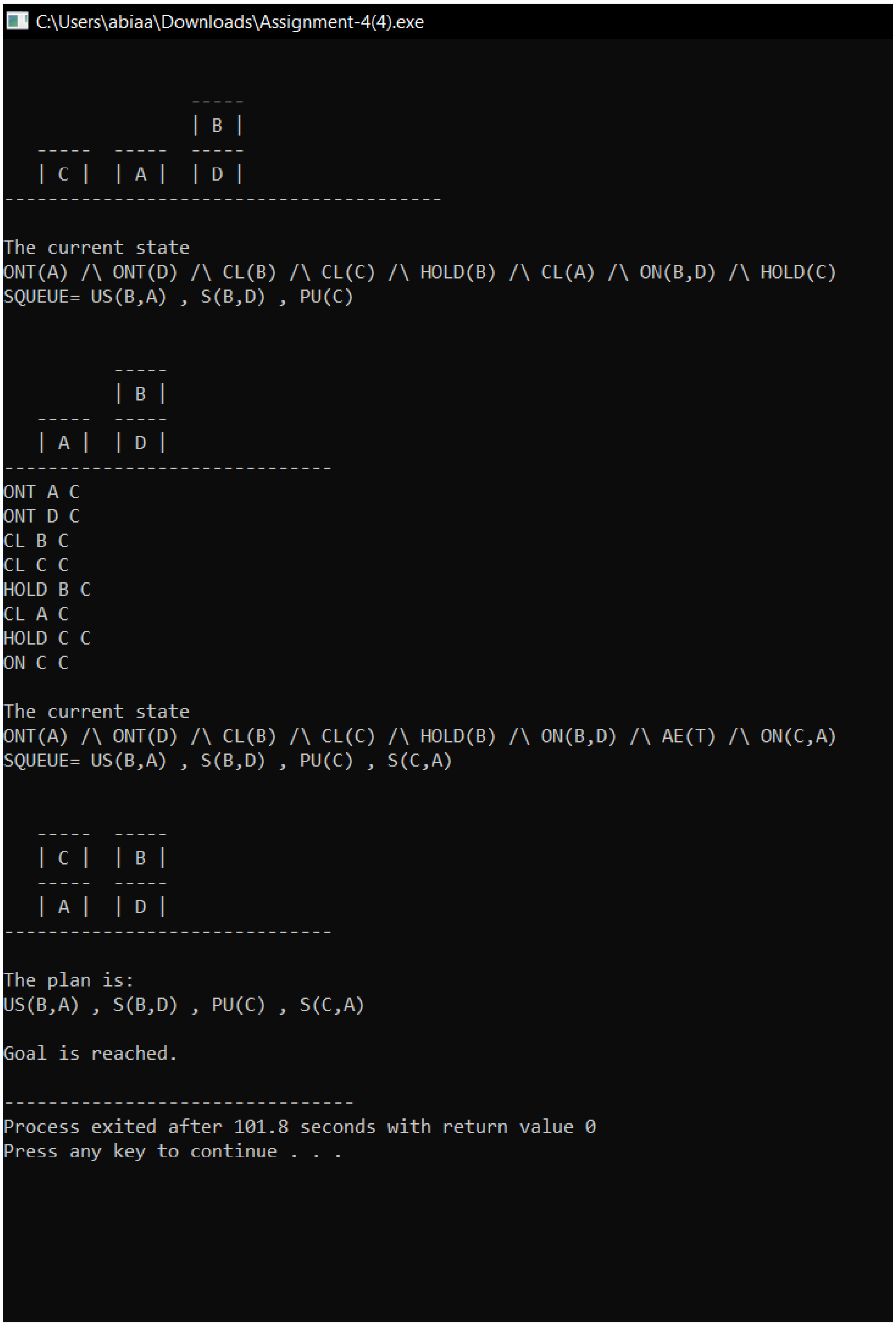


```
C:\Users\abiaa\Downloads\Assignment-4(4).exe
Welcome to Goal Stack Solver
1. ON (Requires two arguments, arg1 ON arg2)
2. CL (Requires one argument, top of arg1 clear)
3. ONT (Requires one argument, arg1 on table)
4. HOLD (Requires one argument, robot arm holding arg1)
5. AE (Requires no argument, robot arm empty)
Anything Else for Exit.....

Enter Start state
Predicate, Argument(s) ?
1 B A
Predicate, Argument(s) ?
3 C
Predicate, Argument(s) ?
3 A
Predicate, Argument(s) ?
3 D
Predicate, Argument(s) ?
2 B
Predicate, Argument(s) ?
2 C
Predicate, Argument(s) ?
2 D
Predicate, Argument(s) ?
5
Predicate, Argument(s) ?
6


Enter Goal state
Predicate, Argument(s) ?
1 C A
Predicate, Argument(s) ?
1 B D
Predicate, Argument(s) ?
3 A
Predicate, Argument(s) ?
3 D
Predicate, Argument(s) ?
2 C
Predicate, Argument(s) ?
2 B
Predicate, Argument(s) ?
5
Predicate, Argument(s) ?
6_
```

Output:



```
C:\Users\abiaa\Downloads\Assignment-4(4).exe
6
Let us begin..............

The start state is
ON(B,A) /\ ONT(C) /\ ONT(A) /\ ONT(D) /\ CL(B) /\ CL(C) /\ CL(D) /\ AE(T)



          -----
          | B |
   -----  -----  -----
   | C |  | A |  | D |
----------------------------------------

The current state
ONT(C) /\ ONT(A) /\ ONT(D) /\ CL(B) /\ CL(C) /\ CL(D) /\ HOLD(B) /\ CL(A)
SQUEUE= US(B,A)





   -----  -----  -----
   | C |  | A |  | D |
----------------------------------------
ONT C B
ONT A B
ONT D B
CL B B
CL C B
CL D B
CL A B
AE T B
ON B B

The current state
ONT(C) /\ ONT(A) /\ ONT(D) /\ CL(B) /\ CL(C) /\ HOLD(B) /\ CL(A) /\ AE(T) /\ ON(B,D)
SQUEUE= US(B,A) , S(B,D)


                 -----
                 | B |
   -----  -----  -----
   | C |  | A |  | D |
----------------------------------------

The current state
ONT(A) /\ ONT(D) /\ CL(B) /\ CL(C) /\ HOLD(B) /\ CL(A) /\ ON(B,D) /\ HOLD(C)
SQUEUE= US(B,A) , S(B,D) , PU(C)


          -----
          | B |
   -----  -----
```



C:\Users\abiaa\Downloads\Assignment-4(4).exe

```


                 -----
                 | B |
   -----  -----  -----
   | C |  | A |  | D |
----------------------------------------

The current state
ONT(A) /\ ONT(D) /\ CL(B) /\ CL(C) /\ HOLD(B) /\ CL(A) /\ ON(B,D) /\ HOLD(C)
SQUEUE= US(B,A) , S(B,D) , PU(C)


          -----
          | B |
   -----  -----
   | A |  | D |
------------------------------
ONT A C
ONT D C
CL B C
CL C C
HOLD B C
CL A C
HOLD C C
ON C C

The current state
ONT(A) /\ ONT(D) /\ CL(B) /\ CL(C) /\ HOLD(B) /\ ON(B,D) /\ AE(T) /\ ON(C,A)
SQUEUE= US(B,A) , S(B,D) , PU(C) , S(C,A)


   -----  -----
   | C |  | B |
   -----  -----
   | A |  | D |
------------------------------

The plan is:
US(B,A) , S(B,D) , PU(C) , S(C,A)

Goal is reached.

--------------------------------
Process exited after 101.8 seconds with return value 0
Press any key to continue . . .
```

Our Implementation:

Explanations for each module of code is provided below:

Predicates:

Predicates are defined under class Pred.

```
class Pred
{
        public:
        char type; //'P' for Pred, 'O' for operator, 'V' for arraylist of preconditions
        string name;
        int num;
        char arg1;
        char arg2;
        vector <Pred> vec;
```

Explanation:
'name'- stores the name of the predicate.
'arg1' - stores the 1st argument of the predicate
'arg2' - stores the 2nd argument of the predicate
'type' - holds 'P' for Pred, 'O' for operator, 'V' for arraylist of preconditions

Initialising Predicates:
Predicates in this program are initialised using the following constructors:

- Default constructor:

```
Pred()
{
        type = '\0';
        arg1 = '\0';
        name = "empty";
        arg2 = '\0';
}
```

- Constructor for predicate that has no arguments.

```
Pred(string n, char t)
{
        type = t;
        arg1 = '\0';
        name = n;
        arg2 = '\0';
}
```

- Constructor for predicate that takes one argument.

```
Pred(string n, char a1, char t)
{
        type = t;
        arg1 = a1;
        name = n;
        arg2 = '\0';
}
```

- Constructor for predicate that takes two arguments.

```
Pred(string n, char a1, char a2, char t)
{
        type = t;
        arg1 = a1;
        arg2 = a2;
        name = n;
}
```

Predicates implemented in the Application:

Predicates used to define the state of the system and their meanings are listed below:

- ON(A, B): block A is on block B.
- CL(A): top of block A is clear.
- ONT(A): block A is on the table.
- HOLD(A): Robot arm is holding block A.

- AE: Robot's arm is empty.

Inputting state:

```
void input(vector <Pred>&L)
{
   int n;
   char arg1, arg2;
   while(n!=6)
   {
          cout<<"Predicate, Argument(s) ? \n";
          cin>>n;
          Pred op;

          switch(n)
          {
                 //ON
                 case 1:
                        cin>>arg1>>arg2;
                        op = Pred("ON",arg1,arg2,'P');
                        L.push_back(op);
                        break;

                 //CL
                 case 2:
                        cin>>arg1;
                        op = Pred("CL",arg1,'P');
                        L.push_back(op);
                        break;

                 //ONT
                 case 3:
                        cin>>arg1;
                        op = Pred("ONT",arg1,'P');
                        L.push_back(op);
                        break;

                 //HOLD
```

```
                case 4:
                        cin>>arg1;
                        op = Pred("HOLD",arg1,'P');
                        L.push_back(op);
                        break;

                //AE
                case 5:
                        op = Pred("AE",'T','P');
                        L.push_back(op);
                        break;

                default:
                        break;
        }
   }
}
```

Explanation:
The 'input' function provides the user with 6 options, of which five are used to describe the initial state of the problem and the sixth is used to indicate the description is over. The switch statement cases and what they represent are listed as follows:

Case 1: ON predicate.
Case 2: CL predicate.
Case 3: ONT predicate.
Case 4: HOLD predicate.
Case 5: AE predicate.
Default: break from switch statement.

Plan of Action:

The plan of action to solve the block word problem is defined under the 'plan' class.

```
class Plan
{
   public:
   vector <Pred>Start, Goal, Curr, plan;
   stack<Pred> goalStack;
```

Explanation:
'vector <Pred>Start, Goal, Curr, plan**;'** defines variables of type vector to maintain the initial state, goal state, current state and plan.

'stack<Pred> goalStack;' defines a stack to maintain goal-stack.

Constructors:

```
Plan(vector <Pred>S,vector <Pred>G)
        {

                Start = S;
                Goal = G;
                Curr = Start;
                make_goalStack();
               display(Start);
        }
```

Creating the goal-stack:

```
        void make_goalStack()
        {
                vector<Pred> tsubg,g;
                int i,j;
                //cout<<"In make goal stack, goal.size= "<<Goal.size()<<"\n";
                for(i=0;i<Goal.size();i++)
                {
                        int flag = 0;
                        for(int j=0;j<Start.size();j++)
                        {
                                flag = 0;
                                if(Goal[i]== Start[j])
                                {
                                        tsubg.push_back(Start[j]);
                                        flag = 1;
                                        break;
                                }
                        }
```

```
                    if(flag == 0)
                    {
                            g.push_back(Goal[i]);
                    }
                    //goalStack.push(Goal[i]);
            }
            for(i=0;i<g.size();i++)
            {
                    goalStack.push(g[i]);
                    //cout<<"pushed in goal stack  "<<g[i].name<<" "<<g[i].arg1<<"
"<<g[i].arg2<<" "<<g[i].type<<"\n";
            }
    }
```

Planning Function:

```
    void func_plan()
    {
            //int tmp=1;
            while(!goalStack.empty())
            {
                    //if (tmp==12)
                    //    break;
                    //tmp++;
                    //cout<<"In func_plan\n";
                    Pred obj = goalStack.top();
                    goalStack.pop();
                    //cout<<"obj popped from goal stack is: "<<obj.name<<" "<<obj.arg1<<"
"<<obj.arg2<<" "<<obj.type<<"\n";
                    if (obj.getType()=='P')
                    {
                            if (find (Curr.begin(), Curr.end(), obj)!= Curr.end()) //found
                                    continue;
                            else //not found
                            {
                    Pred op = operator_for_pred(obj);
                    //add that operator to the goal stack along with its precons
                    if (op.getName()!="empty")
                            push_op(op);
                            }
                    }
                    else if (obj.getType()=='O')
                    {
        //add to plan and change curr state
        //add list to curr state
        op_add_list(obj);
        //del list from curr state
        op_del_list(obj);
        plan.push_back(obj);
        cout<<"\n The current state\n";
        fout<<"\n The current state\n";
        //pr();
                            vector<Pred>::iterator it;
                            for (it=Curr.begin(); it!=Curr.end(); it++)
                            {
                cout << (*it).getName() << "(" << (*it).arg1;
                fout << (*it).getName() << "(" << (*it).arg1;
                                    if ((*it).arg2 != '\0')
                                    {
                                            cout <<"," << (*it).arg2 << ")";
                                            fout <<"," << (*it).arg2 << ")";
                                    }
                                    else
                                    {
                                            cout << ")";
                                            fout << ")";
                                    }
                                    if ((it+1)!=Curr.end())
                                    {
                                            cout<<" /\\ ";
                                            fout<<" /\\ ";
                                    }
                            }
                            cout<<"\n";
                            fout<<"\n";

                            cout<<"SQUEUE= ";
                            fout<<"SQUEUE= ";
                            for (it=plan.begin(); it!=plan.end(); it++)
```

```
                     {
                             //cout<<"In main\n";
                    Pred obj = *it;
                    //cout<<obj.name<<obj.arg1<<obj.arg2<<"\n";
                    if(obj.getType()=='O')
                             {
                    //Pred o1 = obj;
                    cout << obj.getName() << "(" << obj.arg1;
                    fout << obj.getName() << "(" << obj.arg1;
                                  if (obj.arg2 != '\0')
                                  {
                                          cout <<"," << obj.arg2 << ")";
                                          fout <<"," << obj.arg2 << ")";
                                  }
                                  else
                                  {
                                          cout << ")";
                                          fout << ")";
                                  }
                                  if ((it+1)!=plan.end())
                                  {
                                          cout<<" , ";
                                          fout<<" , ";
                                  }
                    }
                     }
                     cout<<"\n";
                     fout<<"\n";
                     display(Curr);
/*
                     cout<<"after pr\n";
vector<Pred>::iterator it;
                     for (it=Curr.begin(); it!=Curr.end(); it++)
                     {
                    //if((*it).getType()=='O')
                             {
                    //Pred o1 = obj;
                    cout << (*it).getName() << "(" << (*it).arg1;
                                  if ((*it).arg2 != '\0')
                                          cout <<"," << (*it).arg2 << ")";
```

```
                                        else cout << ")";
                                        cout<<"\n";
                              }
                              }
                              */
                    }

                    //write the case for  obj being the list of all goals
                     else if(obj.getType()=='V')
                              {
                    vector<Pred> list = obj.vec;
                    vector<Pred>::iterator it;
                    for (it=list.begin();it!=list.end();it++)
                                   {
                         Pred o = *it;
                         if (find (Curr.begin(), Curr.end(), o) == Curr.end())
                                        {
                         goalStack.push(o);
                         }
                    }
                              }
                    }
                    //cout<<"Outside while of func_plan\n";
          }
```

Explanation:
'func_plan' pops the top element of the goal stack, checks the type of element (predicate, operator or preconditions) and proceeds based on type.

Algorithm is as follows:

1. Traverse the goal-stack from top to bottom till empty.
2. For each element popped, check its type:
    a. If type is 'P', check 'find (Curr.begin(), Curr.end(), obj)!= Curr.end()'
       If true, then continue the loop. If false, call 'operator_for_pred' and add that operator to the goal stack along with its preconditions.
    b. If type is 'O', add to plan and change curr state, add list to curr state and delete list from curr state.
    c. If type is 'V', push each individual element of V onto the goal stack.

Push Operator:
Pushes an operator in the goal stack, after which it pushes its precons as well.

List of operators implemented and their meanings are listed below:
US(A,B) - pick up A, which is on B.
S(A,B) - place A on B.
PU(A) - pick up A from table.
PD(X) - put down X on the table.

```
        void push_op(Pred obj)
   {
        //push the operator
        goalStack.push(obj);
        //push its pre-conditions
        if(obj.getType()=='O')
         {
        Pred op2 = obj;
        if(op2.getName()=="US")
                 {
                Pred precon_list("Arraylist", 'V');
                precon_list.vec.push_back(Pred("ON",op2.arg1,op2.arg2,'P'));
                precon_list.vec.push_back(Pred("CL",op2.arg1,'P'));
                precon_list.vec.push_back(Pred("AE",'T','P'));
                goalStack.push(precon_list);
                goalStack.push(Pred("ON",op2.arg1,op2.arg2,'P'));
                goalStack.push(Pred("CL",op2.arg1,'P'));
                goalStack.push(Pred("AE",'T','P'));
        }
        // if op = S
        else if(op2.getName()=="S")
                 {
                Pred precon_list("Arraylist", 'V');
                precon_list.vec.push_back(Pred("CL",op2.arg2,'P'));
                precon_list.vec.push_back(Pred("HOLD",op2.arg1,'P'));
                goalStack.push(precon_list);
                goalStack.push(Pred("CL",op2.arg2,'P'));
                goalStack.push(Pred("HOLD",op2.arg1,'P'));
        }
        // if op = PU
        else if(op2.getName()=="PU")
                {
                Pred precon_list("Arraylist", 'V');
                precon_list.vec.push_back(Pred("ONT",op2.arg1,'P'));
                precon_list.vec.push_back(Pred("AE",'T','P'));
                precon_list.vec.push_back(Pred("CL",op2.arg1,'P'));
                goalStack.push(precon_list);
                goalStack.push(Pred("ONT",op2.arg1,'P'));
                goalStack.push(Pred("AE",'T','P'));
                goalStack.push(Pred("CL",op2.arg1,'P'));
        }
        // if op = PD
        else if(op2.getName()=="PD")
                {
                goalStack.push(Pred("HOLD",op2.arg1,'P'));
        }
        }
    }
```

Add Operator to List:
Checks the type of operator and adds operator list to current state.

```
void op_add_list(Pred obj)
    {
        if(obj.getType()=='O')
         {
        Pred operator1 = obj;
        if(operator1.getName()=="US")
                {
                Curr.push_back(Pred("HOLD",operator1.arg1,'P'));
                Curr.push_back(Pred("CL",operator1.arg2,'P'));
        }
        else if(operator1.getName()=="S")
                {
                Curr.push_back(Pred("AE",'T','P'));
                Curr.push_back(Pred("ON",operator1.arg1,operator1.arg2,'P'));
        }
        else if(operator1.getName()=="PU")
```

```
                {
               Curr.push_back(Pred("HOLD",operator1.arg1,'P'));
        }
        else if(operator1.getName()=="PD")
                {
               Curr.push_back(Pred("AE",'T','P'));
               Curr.push_back(Pred("ONT",operator1.arg1,'P'));
        }
        }
        }
```

Delete operator from list:
Checks the type of operator and deletes operator list from current state.

```
void op_del_list(Pred obj)
   {
        if(obj.getType()=='O')
         {
        Pred operator1 = obj;
        if(operator1.getName()=="US")
                {
               vector<Pred>::iterator it;
               for (it=Curr.begin(); it!=Curr.end(); it++)
                       {
               Pred o = *it;
               if(o.getType()=='P')
                              {
               Pred p = o;
               if(p.getName()=="AE")
                                     {
                       Curr.erase(it); //it.remove();
               }
               else if ((p.getName()=="ON") && (p.arg1 == operator1.arg1) &&
(p.arg2==operator1.arg2))
                                     {
                       Curr.erase(it); //it.remove();
               }
               }
```

```
          }
}
else if(operator1.getName()=="S")
           {
          vector<Pred>::iterator it;
          for (it=Curr.begin(); it!=Curr.end(); it++)
                     {
                               Pred o = *it;
          if(o.getType()=='P')
                               {
          Pred p = o;
          if ((p.getName()=="CL") && (p.arg1==operator1.arg2))
                                         {
                    Curr.erase(it); //it.remove();
          }
          cout<<p.getName()<<" "<<p.arg1<<" "<<operator1.arg1<<endl;
          if ((p.getName()=="HOLD") && (p.arg1==operator1.arg1))
                                         {
                    Curr.erase(it); //it.remove();
          }
          }
          }
}
else if(operator1.getName()=="PU")
           {
          vector<Pred>::iterator it;
          for (it=Curr.begin(); it!=Curr.end(); it++)
                     {
          Pred o = *it;
          if(o.getType()=='P')
                               {
          Pred p = o;
          if ((p.getName()=="ONT") && (p.arg1==operator1.arg1))
                                         {
                    Curr.erase(it); //it.remove();
          }
          if(p.getName()=="AE")
                                         {
                    Curr.erase(it); //it.remove();
          }
```

```
                                }
                        }
                }
                else if(operator1.getName()=="PD")
                         {
                        vector<Pred>::iterator it;
                        for (it=Curr.begin(); it!=Curr.end(); it++)
                                 {
                        Pred o = *it;
                        if(o.getType()=='P')
                                        {
                        Pred p = o;
                        cout<<p.getName()<<" "<<p.arg1<<" "<<operator1.arg1<<operator1.arg1<<endl;
                        if((p.getName()=="HOLD") && (p.arg1==operator1.arg1))
                                                 {
                                Curr.erase(it); //it.remove();
                        }
                        }
                        }
                }
                }
                }
```

Operator for Pred:

```
Pred operator_for_pred(Pred pred)
   {
        Pred  opfinal;
        if(pred.getType()=='P')
         {
        Pred p = pred;
        //ae
        if(p.getName()=="AE")
                 {
                char x = 0;
                vector<Pred>::iterator it;
                for (it=Curr.begin(); it!=Curr.end(); it++)
                         {
                Pred o = *it;
        if(o.getType()=='P')
                  {
                        Pred p1 = o;
        if(p1.getName()=="HOLD")
                        {
            x = p1.arg1;
        }
        }
        }
        Pred op("PD", x,'O');
        opfinal = op;
}
//hold
else if(p.getName()=="HOLD")
         {
        char x = '\0';
        vector<Pred>::iterator it, it2;
        //if ont and cl then pu
        for (it=Curr.begin(); it!=Curr.end(); it++)
                 {
        Pred o = *it;
        if(o.getType()=='P')
                  {
        Pred o2 = o;
        if ((o2.getName()=="ONT") && (o2.arg1 == p.arg1))
                        {
            //check if CL is also true
                for (it2=Curr.begin(); it2!=Curr.end(); it2++)
                              {
            Pred o_ = *it2;
            if(o_.getType()=='P')
                                    {
            Pred o2_ = o_;
            if((o2_.getName()=="CL") && (o2_.arg1 == p.arg1))
                                          {
                Pred op1("PU",p.arg1,'O');
                opfinal = op1;
            }
            }
            }
        }
        }
        }

        //if on something then US
        vector<Pred>::iterator i;
        //if ont and cl then pu
        for (i=Curr.begin(); i!=Curr.end(); i++)
                 {
        Pred o = *i;
        if(o.getType()=='P')
                                {
        Pred o2 = o;
        if ((o2.getName()=="ON") && (o2.arg1 == p.arg1))
                                            {
            x = o2.arg2;
            Pred op2 ("US",p.arg1, x,'O');
            opfinal = op2;
        }
        }
        }
}
//cl
else if(p.getName()=="CL")
         {
        char x = '\0';
        //find x
        vector<Pred>::iterator it;
        for (it=Curr.begin(); it!=Curr.end(); it++)
                 {
        Pred o = *it;
        if(o.getType()=='P')
                                {
        Pred o2 = o;
        if ((o2.getName()=="ON") && (o2.arg2 == p.arg1))
                                            {
            x = o2.arg1;
        }
        }
        }
```

```
                        if (x!='\0')
                        {
                Pred op ("US",x,p.arg1,'O');
                opfinal = op;
                        }
        }
        //ont
        else if(p.getName()=="ONT")
                 {
                Pred op ("PD",p.arg1,'O');
                opfinal = op;
        }

        else if (p.getName()=="ON")
                 {
                //on
                Pred p = pred;
                Pred op("S",p.arg1,p.arg2,'O');
                opfinal = op;
        }
        }
        return opfinal;
        }
```

Explanation:
The ‘operator_for_pred’ function checks the type of predicate by getting the Predicate name. Based on the name of the predicate, the function returns the appropriate operator, which is stored in the variable ‘opfinal’.

Display:
A function that displays the state of the system in an organised manner.

```
        void display(vector <Pred>disp)
        {
                 int len = disp.size(),maxR = 0,maxC = 1,i,j;
                 char draw[len+1][len+1];
                 for(i=0;i<=len;i++)
                 {
                        for(j=0;j<=len;j++)
                        {
                                draw[i][j] = '\0';
                        }
                }
                cout<<"\n\n";
                vector<Pred>::iterator it;
        for (it=disp.begin(); it!=disp.end(); it++)
                {
                        if((*it).name == "ONT")
                        {
                                draw[maxR][0] = (*it).arg1;
                                maxR++;
                        }
                }
                for (it=disp.begin(); it!=disp.end(); it++)
                {
                        int found=0;
                        if((*it).name == "ON")
                        {
                                for(i=0;i<maxR;i++)
                                {
                                        for(j=0;j<maxC;j++)
                                        {
                                                if(draw[i][j] == (*it).arg2)
                                                {
                                                        draw[i][j+1] = (*it).arg2;
                                                        if((j+1)>maxC)
                                                        {
                                                                maxC = j+1;
                                                                found=1;
                                                                break;
                                                        }
                                                }
                                        }
                                        if (found==1)
                                                break;
                                }
                        }
                }
                for(j=maxC;j>=0;j--)
        {
            string str1,str2,str3;
            for(i=maxR;i>=0;i--)
            {
                if(draw[i][j] == '\0')
                {
                    cout<<"        ";
                }
                else
                {
                    cout<<"___\n";
                    cout<<draw[j][i];
                }
            }
            cout<<"\n";
        }
        for(i=0;i<maxR;i++)
        {
            cout<<"_";
        }
        cout<<"\n\n\n";
    }
```

Main function:

```
int main()
{
   vector<Pred> S, G;
   cout<<"Welcome to Goal Stack Solver\n";
   cout<<"1. ON (Requires two arguments, arg1 ON arg2)\n";
   cout<<"2. CL (Requires one argument, top of arg1 clear)\n";
   cout<<"3. ONT (Requires one argument, arg1 on table)\n";
   cout<<"4. HOLD (Requires one argument, robot arm holding arg1)\n";
cout<<"5. AE (Requires no argument, robot arm empty)\n";
cout<<"Anything Else for Exit.....\n\n";
cout<<"Enter Start state\n";
input(S);

cout<<"\n\nEnter Goal state\n";
input(G);
//print the first current state as the start state

cout<<"Let us begin..............\n\n";
fout<<"Let us begin..............\n\n";
cout<<"The start state is\n";
fout<<"The start state is\n";
vector<Pred>::iterator it1 ;
        for (it1=S.begin(); it1!=S.end(); it1++)
{
        cout << (*it1).getName() << "(" << (*it1).arg1;
        fout << (*it1).getName() << "(" << (*it1).arg1;
         if ((*it1).arg2 != '\0')
         //if ((*it1).name == "ON")
         {
                    cout <<"," << (*it1).arg2 << ")";
                    fout <<"," << (*it1).arg2 << ")";
         }
         else
         {
                    cout << ")";
                    fout << ")";
         }
         if ((it1+1)!=S.end())
         {
                    cout<<" /\\ ";
                    fout<<" /\\ ";
         }
}
cout<<"\n";
fout<<"\n";

Plan P(S, G);
        P.func_plan();
        vector<Pred> plan = P.plan;
        //cout<<"In main\n";
        vector<Pred>::iterator it ;
   cout<<"\n \n The plan is: \n";
   fout<<"\n \n The plan is: \n";
   for (it=plan.begin(); it!=plan.end(); it++)
   {
         //cout<<"In main\n";
        Pred obj = *it;
        //cout<<obj.name<<obj.arg1<<obj.arg2<<"\n";
        if(obj.getType()=='O')
         {
        //Pred o1 = obj;
        cout << obj.getName() << "(" << obj.arg1;
        fout << obj.getName() << "(" << obj.arg1;
                if (obj.arg2 != '\0')
                {
                        cout <<"," << obj.arg2 << ")";
                        fout <<"," << obj.arg2 << ")";
                }
                else
                {
                        cout << ")";
                        fout << ")";
                }
                if ((it+1)!=plan.end())
                {
                        cout<<" , ";
                        fout<<" , ";
                }
        }
   }
   cout<<"\n";
   fout<<"\n";
   cout<<"Goal is reached. \n";
   fout<<"Goal is reached. \n";
   return 0;
}
```

Explanation:

The ‘main’ function displays a menu indicating the options and the predicates they are matched to. It then calls the ‘input’ function to read the initial and goal states, after which it calls the ‘plan’ function to solve the word block problem using the goal stack method.