Usage: ./yasat <filename>

and the answer <filename>.sat will be in the same directory.

Workflow

the structure is based on the vanilla DPLL algorithm from lectures.

- **DPLL(set_of_clauses)**

```
// do BCP
while (set_of_clauses contains a unit clause due to literal L) {
    Simplify set_of_clauses by setting variable for L to its required value in all clauses
}
If (set_of_clauses is all "1" clauses now)
    return (SAT)  // you have simplified every clause to be "1"
if (set_of_clauses contains a clause that evals to "0")
    return (UNSAT) // this is a conflict, this set of var assignment doesn't satisfy
//  must recurse
Heuristically choose an unassigned variable x and heuristically choose a value v
    if( DPLL(set_of_clauses = simplified by setting x=v) == SAT )
      return (SAT)
    else return ( DPLL(set_of_clauses = simplified by setting x= ¬v) )
```

Variable

having three major arrays to hold the information needed.

X : x1...xn values, default set to -1

cl_val[i] : bool value for clause i, default set to -1

cl_omega[i] : number of unassigned variable at clause i, default to original size.

Branching heuristic

using MOM(Most Often in Minimal) method, which picking the most often unassigned literal in clauses where cl_imega[i] is smaller than threshold (default 3).

Summary of result

The test data in benchmark should be all satisfied, though some cnf have more than one solution.

Difficulties

1. Need restoring the change of implication after detect UNSAT
    a. My solution now is recalculating whole clauses again, which is time-consuming. I will design better data structure and refractor the code till next milestone.
2. The implication itself need recursion
    a. I use a integer to record the literal assigned after calling *make_imlication(),* if old_num == new_num, meaning this implication reach end.