Experiment No : 5

Aim : Write a C/C++ program to read a Well Formed Formula(wff) in Propositional Logic and determine whether a given wff is satisfiable or valid.

Theory :
Propositional Logic:
Propositional logic, also known as sentential logic and statement logic, is the branch of logic that studies ways of joining and/or modifying entire propositions, statements or sentences to form more complicated propositions, statements or sentences, as well as the logical relationships and properties that are derived from these methods of combining or altering statements.
Well Formed Formula(WFF):
Any expression that obeys the syntactic rules of propositional logic is called a well-formed formula, or WFF. These rules are:
1. Any capital letter by itself is a WFF.
2. Any WFF can be prefixed with . (The result will be a WFF too.)
3. Any two WFFs can be put together with one of ·, ∨, ⊃, or ≡ between them, enclosing the result in parentheses. (This will be a WFF too.)

Code :
```
#include<iostream>
#include<bitset>
#include<string>
#include<stack>
#include<list>
#include<vector>
#include<algorithm>
#include<map>
using namespace std;
#define MAX_EXPR_SIZE 100
int numVars;
map<string,int> varPos; // Position of a variable in bitset
typedef enum TokenType{
VAR,
AND,
OR,
NOT,
OPEN_PAREN,
CLOSE_PAREN,
}TokenType;
int precendence(TokenType t){
switch(t){
case AND:
case OR:
return 1;
case NOT:
return 2;
default:
return -1;
}
}
```

```
typedef struct Token{
string name;
TokenType type;
bool value;
Token(){name="",type=VAR;}
Token(string v,TokenType t):name(v),type(t){}
Token(TokenType t,bool v):name(""),type(t),value(v){}
void setValue(bitset<1+MAX_EXPR_SIZE/2> values){
value=values[varPos[name]];
}
}Token;
typedef struct valueNode{
bitset<1+MAX_EXPR_SIZE/2> varValues;
int idx;
}node;
bool isOp(char x){
switch(x){
case '&':
case '|':
case '~':
case ')':
case '(':
return true;
default:
return false;
}
}
bool cmpVars(pair<string,int>& a, pair<string,int>& b){
return a.second< b.second;
}
void printStack(stack<Token> t){
while(!t.empty()){
cout<<t.top().name<<endl;
t.pop();
}
}
void printTableHeader(string inputExp){
vector<pair<string,int>> variables;
for(auto pairs: varPos)
variables.push_back(pairs);
sort(variables.begin(),variables.end(),cmpVars);
for(auto pairs:variables)
cout<<pairs.first<<" ; ";
cout<<inputExp<<endl;
}
void printValues(bitset<1+MAX_EXPR_SIZE/2> values, bool result){
int i;
for(i=0;i<numVars;i++){
cout<<values[i]<<" ; ";
}
cout<<" "<<result<<endl;
```

```
return;
}
inline void parseBinaryOp(int idx, stack<Token> *postfix,stack<Token> *op, string
input){
while(!op->empty() && precendence(op->top().type)>=1){
postfix->push(op->top());
op->pop();
}
op->emplace(input.substr(idx,1),input[idx]=='&' ? AND:OR);
// check for correctness of input
if(input[idx+1]!='~' && isOp(input[idx+1])){
cout<<"Error at operator "<<idx+1<<" i.e. "<<input[idx+1];
cout<<" given expression is not well formed"<<endl;;
}
}
inline int parseVariable(int idx, stack<Token>* postfix,string input){
int x=0;
while(idx+x<input.length() && !isOp(input[idx+x])){
x++;
}
string var=input.substr(idx,x);
postfix->push({var,VAR});
if(varPos.find(var)==varPos.end()){
varPos[var]=numVars;
numVars++;
}
idx+=x-1;
// check for correctness of input
if(input[idx+1]=='~'){
cout<<"Error at operator "<<idx+1<<" i.e. "<<input[idx+1];
cout<<" given expression is not well formed"<<endl;;
exit(1);
}
return idx;
}
stack<Token> infixToPostfix(string input){
stack<Token> op,postfix;
int idx=0;
while(idx<input.length()){
switch(input[idx]){
// parenthesis
case '(':
op.emplace("(",OPEN_PAREN);
break;
case ')':
while(!op.empty() && op.top().type!=OPEN_PAREN){
postfix.push(op.top());
op.pop();
}
if(!op.empty())
op.pop();
```

```
break;
// binary operators
case '&':
case '|':
parseBinaryOp(idx,&postfix,&op,input);
break;
// unary operator
case '~':
while(!op.empty() && precendence(op.top().type)>=2){
postfix.push(op.top());
op.pop();
}
op.emplace(input.substr(idx,1),NOT);
break;
default:
//variable
idx=parseVariable(idx, &postfix, input);
break;
}
idx++;
}
// empty the operator stack
while(!op.empty()){
postfix.push(op.top());
op.pop();
}
//reverse postfix
stack<Token> temp;
while(!postfix.empty()){
temp.push(postfix.top());
postfix.pop();
}
if(temp.size()==0){
cout<<"Error: No tokens found"<<endl;
exit(1);
}
return temp;
}
bool evaluate(stack<Token> postfix,bitset<1+MAX_EXPR_SIZE/2> values){
stack<Token> result;
Token a,b,op;
while(!postfix.empty()){
if(postfix.top().type==VAR){
postfix.top().setValue(values);
result.push(postfix.top());
postfix.pop();
continue;
}
a=result.top();
result.pop();
switch(postfix.top().type){
```

```
case AND:
b=result.top();
result.pop();
result.emplace(VAR,a.value&b.value);
break;
case OR:
b=result.top();
result.pop();
result.emplace(VAR,a.value|b.value);
break;
case NOT:
result.emplace(VAR,!a.value);
break;
default:
cout<<"Error: expected operator token, got: "<<postfix
.top().name;
cout<<" with name:"<<postfix.top().name<<endl;
exit(1);
}
postfix.pop();
}
printValues(values,result.top().value);
return result.top().value;
}
void generateTable(stack<Token> e,string inputExp){
/* Generates the truth table;*/
printTableHeader(inputExp);
int totalEntries=0, totalTrue=0,i;
node start={.idx=-1},tmp;
bool result;
list<node> table;
totalEntries++;
if(evaluate(e,start.varValues))
totalTrue++;
table.push_front(start);
int count=0;
while(!table.empty()){
tmp=table.front();
table.pop_front();
i=tmp.idx+1;
if (i>=numVars){
continue;
}
table.emplace_back(node{tmp.varValues,i });
tmp.varValues[i]=1;
table.emplace_back(node{tmp.varValues,i });
if(evaluate(e,tmp.varValues)){
totalTrue++;
}
totalEntries++;
}
```

```
cout<<"entries: "<<totalEntries<<endl;
cout<<"total true: "<<totalTrue<<endl;
cout<<endl<<"Therefore the given wff is ";
if(totalTrue==totalEntries)
cout<<"valid"<<endl;
else if(totalTrue==0)
cout<<"unsatisfiable"<<endl;
else
cout<<"satisfiable"<<endl;
}
int main(){
string input;
cout<<"Enter expression to be evalutated: ";
cin>>input;
stack<Token> postfix=infixToPostfix(input);
generateTable(postfix,input);
}
```

Output :

```
1. Enter expression to be evalutated: l|m(g)
l ; m ; g ; l|m(g)
0 ; 0 ; 0 ;   0
1 ; 0 ; 0 ;   0
0 ; 1 ; 0 ;   1
1 ; 1 ; 0 ;   1
0 ; 0 ; 1 ;   1
0 ; 1 ; 1 ;   1
1 ; 0 ; 1 ;   1
1 ; 1 ; 1 ;   1
entries: 8
total true: 6
Therefore the given wff is satisfiable

2. Enter expression to be evalutated: l&~l
l ; l&~l
0 ;   0
1 ;   0
entries: 2
total true: 0
Therefore the given wff is unsatisfiable

3. Enter expression to be evalutated: m|~m
m ; m|~m
0 ;   1
1 ;   1
entries: 2
total true: 2
Therefore the given wff is valid
```

Conclusion : The given program was successfully written and executed.

Deepraj Bhosale
181105016