```
// Copyright 2002 - www.karamian.com
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <fstream.h>
#include <string.h>


/* Structs */

typedef struct varTableStruct
{
    char varName[10];
    int  value;
} varTableStruct;

typedef struct condStruct
{
    char var[10];
    int  set;
    int  equal;
} condStruct;


typedef struct jobStruct
{
    char var[10];
    int  set;
} jobStruct;

typedef struct instruction
{
    condStruct cond;
    condStruct cond2;
    int andor;

    jobStruct  job;
} instruction;
```

```
typedef struct process {
    instruction ins[50];
    int pointer;
    int our;
    int other;
} process;


int getRand();
int readSolution(ifstream& data, process& producer, process& consumer);
void runsolution(process producer, process consumer,int& isMutualEx,int& isDeadLock);
void setSolutionDefaults(process& producer, process& consumer);
void setVarDefaults(process& producer, process& consumer);
void runInstruction(instruction ins, int& pointer, process proc);

int  getVar(char* name);
varTableStruct*  getVarTable(char* name);

varTableStruct varTable[] = {
    { "prod_lock" , 0 },
    { "cons_lock" , 0 },
    { "ts"        , 0 },
    { "true"      , 1 },
    { "size"      , 3 },
    { "flag"      , 0 },
    { "flag1"     , 0 },
    { "flag2"     , 0 },
    { "turn"      , 1 },
    { "accu"      , 0 },
    { "jump"      , 0 },
    { "end"       , 0 }
};



char* strrsts(char* str, char* sub)
{
    char* addr;
    for (int i=0 ; i< strlen(sub); i++)
```

```
    {
        addr = strrchr(str, sub[i]);
        if (!addr)
            return NULL;
        if (i>0)
        {
            if (str[addr- str -1] != sub[i-1])
                return NULL;
        }
    }

    return addr;
}



int main()
{
    process producer, consumer;

    /* Set the seed */
    srand( (unsigned) time(NULL));

    ifstream data("/dfs/user/ashirdel/cs431/Csp.dat");



    int solution = 1;
    while (readSolution(data, producer, consumer))
    {
        cout << "\nSOLUTION " << solution++ << endl;
        int isMutualEx=0, isDeadLock=0;
        for (int s=0; s<50;s++)
        {
            setVarDefaults(producer,consumer);
            cout << "RUN" << s+1 << "      " ;
            runsolution(producer, consumer, isMutualEx, isDeadLock);
        }

        if (isMutualEx)
            cout << "*** VIOLATE MUTUAL EXCLUSION  ***" << endl;
        if (isDeadLock)
```

```
            cout << "*** VIOLATE BOUNDED WAIT  ***" << endl;
        if (!isMutualEx && !isDeadLock)
            cout << "*** CORRECT SOLUTION  ***" << endl;

    }

    return 0;
}

int getRand()
{
    if ( rand() < RAND_MAX/2 )
        return 0;
    else
        return 1;
}

void setSolutionDefaults(process& producer, process& consumer)
{
    for (int c=0; c<50; c++)
    {
        strcpy(producer.ins[c].cond.var,"true");
        producer.ins[c].cond.set = 1;
        producer.ins[c].cond.equal = 1;
        strcpy(consumer.ins[c].cond.var,"true");
        consumer.ins[c].cond.set = 1;
        consumer.ins[c].cond.equal = 1;

        strcpy(producer.ins[c].cond2.var,"true");
        producer.ins[c].cond2.set = 1;
        producer.ins[c].cond2.equal = 1;
        strcpy(consumer.ins[c].cond2.var,"true");
        consumer.ins[c].cond2.set = 1;
        consumer.ins[c].cond2.equal = 1;

        producer.ins[c].andor = 0;
        consumer.ins[c].andor = 0;
    }
```

```
    producer.our = 1;
    producer.other = 2;

    consumer.our = 2;
    consumer.other = 1;

}

void setVarDefaults(process& producer, process& consumer)
{
    varTable[4].value = 3;
    varTable[5].value = 0;
    varTable[6].value = 0;
    varTable[7].value = 0;
    varTable[8].value = 1;
    varTable[9].value = 0;
}

int readSolution(ifstream& data, process& producer, process& consumer)
{
    char temp[64];
    char job[64];
    char cond[64];

    data.getline(temp, 64);

    if ( strncmp(temp, "end", 3) == 0)
        return 0;

    if ( strncmp(temp, "solution", 8) != 0)
    {
        cout << "error in file" << endl;
        exit(1);
    }

    setSolutionDefaults(producer,consumer);
    int ins = 0;
    while (1)
    {
        data.getline(temp, 64);
```

```
if (strncmp(temp, "end",3) == 0)
    break;

char* addr = strrchr(temp, ',');
if (!addr)
{
    cout << "error in file, line = "<< temp << endl;
    exit(1);
}

strncpy(job, temp, (size_t) (addr - temp));
job[(addr - temp)] = 0;
strcpy(cond, (const char*) (addr + 1));

/* Pars JOBs */
if (strncmp(job, "lock", 4) == 0)
{
    strcpy(producer.ins[ins].job.var,"prod_lock");
    producer.ins[ins].job.set = 1;

    strcpy(consumer.ins[ins].job.var,"cons_lock");
    consumer.ins[ins].job.set = 1;
}
else if (strncmp(job, "ts", 4) == 0)
{
    strcpy(producer.ins[ins].job.var,"ts");
    producer.ins[ins].job.set = 1;

    strcpy(consumer.ins[ins].job.var,"ts");
    consumer.ins[ins].job.set = 1;
}
else if (strncmp(job, "cs", 2) == 0)
{
    strcpy(producer.ins[ins].job.var,"jump");
    producer.ins[ins].job.set = 0;

    strcpy(consumer.ins[ins].job.var,"jump");
    consumer.ins[ins].job.set = 0;
    ins++;
```

```
    strcpy(producer.ins[ins].job.var,"accu");
    producer.ins[ins].job.set = 4;

    strcpy(consumer.ins[ins].job.var,"accu");
    consumer.ins[ins].job.set = 4;
    ins++;

    strcpy(producer.ins[ins].job.var,"accu");
    producer.ins[ins].job.set = 2;

    strcpy(consumer.ins[ins].job.var,"accu");
    consumer.ins[ins].job.set = 3;
    ins++;

    strcpy(producer.ins[ins].job.var,"size");
    producer.ins[ins].job.set = 5;

    strcpy(consumer.ins[ins].job.var,"size");
    consumer.ins[ins].job.set = 5;
}
else
{
    char var[10];
    char* varAddr = strrchr(job, '=');
    if (!varAddr)
    {
        cout << "error in file, job = "<< job << endl;
        exit(1);
    }
    strncpy(var, job, (size_t) (varAddr - job));
    var[varAddr - job] = 0;

    strcpy(producer.ins[ins].job.var,var);
    strcpy(consumer.ins[ins].job.var,var);

    int set;
    if (strncmp((char*)(varAddr + 1), "our",3) == 0)
        set = 6;
    else
```

```
    if (strncmp((char*)(varAddr + 1), "other",5) == 0)
        set = 7;
    else
        set =  *((char*)(varAddr + 1)) - '0';
    producer.ins[ins].job.set = set;
    consumer.ins[ins].job.set = set;

    varAddr = strrchr(var, '_');
    if (varAddr)
    {
        if (strncmp(((char*)(varAddr + 1)),"our",3) == 0)
        {
            varAddr[0] = '1';
            varAddr[1] = 0;
            strcpy(producer.ins[ins].job.var,var);

            varAddr[0] = '2';
            varAddr[1] = 0;
            strcpy(consumer.ins[ins].job.var,var);
        }

        if (strncmp(((char*)(varAddr + 1)),"other",5) == 0)
        {
            varAddr[0] = '2';
            varAddr[1] = 0;
            strcpy(producer.ins[ins].job.var,var);

            varAddr[0] = '1';
            varAddr[1] = 0;
            strcpy(consumer.ins[ins].job.var,var);
        }
    }
}

/* Pars Conditions */
if (strncmp(cond, "1", 1) == 0)
{
    strcpy(producer.ins[ins].cond.var,"true");
    producer.ins[ins].cond.set = 1;
```

```
    strcpy(consumer.ins[ins].cond.var,"true");
    consumer.ins[ins].cond.set = 1;
}
else
{
    int offset = 0;
    char var[10];

    char* varAddr = strchr(cond, '=');

    if (!varAddr)
    {
        cout << "error in file, cond = "<< cond << endl;
        exit(1);
    }

    char* notEqual= strchr(cond, '!');

    if (notEqual && (notEqual == (varAddr - 1)))
    {
        producer.ins[ins].cond.equal = 0;
        consumer.ins[ins].cond.equal = 0;
        offset = 1;
    }

    strncpy(var, cond, (size_t) (varAddr - cond));
    var[varAddr - cond - offset] = 0;

    strcpy(producer.ins[ins].cond.var,var);
    strcpy(consumer.ins[ins].cond.var,var);

    int set;
    if (strncmp((char*)(varAddr + 1), "our",3) == 0)
        set = 6;
    else
    if (strncmp((char*)(varAddr + 1), "other",5) == 0)
        set = 7;
    else
        set =  *((char*)(varAddr + 1)) - '0';
```

```
producer.ins[ins].cond.set = set;
consumer.ins[ins].cond.set = set;

varAddr = strchr(var, '_');
if (varAddr)
{
    if (strncmp(((char*)(varAddr + 1)),"our",3) == 0)
    {
        varAddr[0] = '1';
        varAddr[1] = 0;
        strcpy(producer.ins[ins].cond.var,var);

        varAddr[0] = '2';
        varAddr[1] = 0;
        strcpy(consumer.ins[ins].cond.var,var);
    }

    if (strncmp(((char*)(varAddr + 1)),"other",5) == 0)
    {
        varAddr[0] = '2';
        varAddr[1] = 0;
        strcpy(producer.ins[ins].cond.var,var);

        varAddr[0] = '1';
        varAddr[1] = 0;
        strcpy(consumer.ins[ins].cond.var,var);
    }
}


char* andor = strrchr(cond, '|');
if (andor)
{
    consumer.ins[ins].andor =1;
    producer.ins[ins].andor =1;
}
else
{
    andor = strchr(cond, '&');
```

```
}

if (andor)
{
    offset =0;
    char* cond2 = (char*) (andor + 1);
    char* varAddr = strchr(cond2, '=');

    if (!varAddr)
    {
        cout << "error in file, cond2 = "<< cond2 << endl;
        exit(1);
    }

    char* notEqual= strchr(cond2, '!');

    if (notEqual && (notEqual == (varAddr - 1)))
    {
        producer.ins[ins].cond2.equal = 0;
        consumer.ins[ins].cond2.equal = 0;
        offset = 1;
    }

    strncpy(var, cond2, (size_t) (varAddr - cond2));
    var[varAddr - cond2 - offset] = 0;

    strcpy(producer.ins[ins].cond2.var,var);
    strcpy(consumer.ins[ins].cond2.var,var);

    int set;
    if (strncmp((char*)(varAddr + 1), "our",3) == 0)
        set = 6;
    else
    if (strncmp((char*)(varAddr + 1), "other",5) == 0)
        set = 7;
    else
        set =  *((char*)(varAddr + 1)) - '0';

    producer.ins[ins].cond2.set = set;
    consumer.ins[ins].cond2.set = set;
```

```
                varAddr = strchr(var, '_');
                if (varAddr)
                {
                    if (strncmp(((char*)(varAddr + 1)),"our",3) == 0)
                    {
                        varAddr[0] = '1';
                        varAddr[1] = 0;
                        strcpy(producer.ins[ins].cond2.var,var);

                        varAddr[0] = '2';
                        varAddr[1] = 0;
                        strcpy(consumer.ins[ins].cond2.var,var);
                    }

                    if (strncmp(((char*)(varAddr + 1)),"other",5) == 0)
                    {
                        varAddr[0] = '2';
                        varAddr[1] = 0;
                        strcpy(producer.ins[ins].cond2.var,var);

                        varAddr[0] = '1';
                        varAddr[1] = 0;
                        strcpy(consumer.ins[ins].cond2.var,var);
                    }
                }

            }
        }

        ins++;
    }
    producer.ins[ins].job.var[0] = NULL;
    consumer.ins[ins].job.var[0] = NULL;
    return 1;
}

int  getVar(char* name)
{
```

```
    int c=0;

    do
    {
        if (strncmp(varTable[c].varName, "end", 3) == 0)
            return -1;
        if (strcmp(name,varTable[c].varName) == 0)
            return varTable[c].value;
    } while(++c);

    return -1;
}

varTableStruct* getVarTable(char* name)
{
    int c=0;

    do
    {
        if (strncmp(varTable[c].varName, "end", 3) == 0)
            return NULL;
        if (strcmp(name,varTable[c].varName) == 0)
            return &varTable[c];
    } while (++c);

    return NULL;
}

void runsolution(process producer, process consumer,int& isMutualEx,int& isDeadLock)
{
    int counter = 0;
    producer.pointer = consumer.pointer = 0;

    while ( (producer.pointer >=0 || consumer.pointer >=0) &&
            counter <= 1000)
    {
        if (getRand())
        {
            if (producer.pointer >=0)
            {
```

```
                runInstruction(producer.ins[producer.pointer],
                    producer.pointer, producer);
                counter ++;
            }
        }
        else
        {
            if (consumer.pointer >= 0)
            {
                runInstruction(consumer.ins[consumer.pointer],
                    consumer.pointer, consumer);
                counter ++;
            }
        }
    }

    if (counter >=1000)
        isDeadLock = 1;
    else if (getVar("size") != 3)
        isMutualEx = 1;

    cout << "SIZE = " <<  getVar("size") << "      "
         << "COUNTER = "<< counter << endl;

}

void runInstruction(instruction ins, int& pointer, process proc)
{
    varTableStruct* tbl;

    if (ins.job.var[0] == 0)
    {
        pointer = -1;
        return;
    }

    int condition1 = 0 , condition2 = 0;

    if (ins.cond.equal == 1)
```

```
{
    if (ins.cond.set == 6 && getVar(ins.cond.var) == proc.our)
    condition1 =1;
    else if (ins.cond.set == 7 && getVar(ins.cond.var) == proc.other)
    condition1 =1;
    else if (getVar(ins.cond.var) == ins.cond.set)
        condition1 = 1;
}
else
{
    if (ins.cond.set == 6 && getVar(ins.cond.var) != proc.our)
    condition1 =1;
    else if (ins.cond.set == 7 && getVar(ins.cond.var) != proc.other)
    condition1 =1;
    else if ((ins.cond2.set <3) && (getVar(ins.cond.var) != ins.cond.set))
        condition1 = 1;
}

if (ins.cond2.equal == 1)
{
    if (ins.cond2.set == 6 && getVar(ins.cond2.var) == proc.our)
    condition2 =1;
    else if (ins.cond2.set == 7 && getVar(ins.cond2.var) == proc.other)
    condition2 =1;
    else if (getVar(ins.cond2.var) == ins.cond2.set)
        condition2= 1;
}
else
{
    if (ins.cond2.set == 6 && getVar(ins.cond2.var) != proc.our)
    condition2 =1;
    else if (ins.cond2.set == 7 && getVar(ins.cond2.var) != proc.other)
    condition2 =1;
    else if ( (ins.cond2.set <3) && (getVar(ins.cond2.var) != ins.cond2.set))
        condition2= 1;
}

if ( (ins.andor && (condition1 || condition2)) ||
     (!ins.andor && condition1 && condition2))
{
```

```
    tbl = getVarTable(ins.job.var);

    if (!tbl)
    {
        cout << "error, job variable does not exist\n";
        exit(1);
    }

    if (ins.job.set == 2)
        tbl->value ++;
    else if (ins.job.set == 3)
        tbl->value --;
    else if (ins.job.set == 4)
        tbl->value = getVar("size");
    else if (ins.job.set == 5)
        tbl->value = getVar("accu");
    else if (ins.job.set == 6)
        tbl->value = proc.our;
    else if (ins.job.set == 7)
        tbl->value = proc.other;
    else
        tbl->value = ins.job.set;

    if (strcmp(tbl->varName, "prod_lock") ==0 ||
        strcmp(tbl->varName, "cons_lock") ==0 ||
        strcmp(tbl->varName, "ts") == 0)
    {
        pointer --;
    }

}
else
if (strcmp(ins.job.var, "ts") ==0)
{
    tbl = getVarTable(ins.cond.var);
    if (!tbl)
    {
        cout << "error, job variable does not exist\n";
```

```
            exit(1);
        }
        tbl->value = ins.cond.set;
    }

    pointer ++;
}

// Data File. Csp.dat

solution 1
lock,flag=1
flag=1,1
cs,1
flag=0,1
end
solution 2
flag_our=1,1
lock,flag_other=1
cs,1
flag_our=0,1
end
solution 3
lock,flag_other=1
flag_our=1,1
cs,1
flag_our=0,1
end
solution 4
flag_our=1,1
lock,flag_other=1 &turn!=our
cs,1
turn=other,1
flag_our=0,1
end
solution 5
flag_our=1,1
turn=other,1
lock,flag_other=1 &turn!=our
cs,1
flag_our=0,1
end
solution 6
ts,flag=1
cs,1
flag=0,1
end
end
```