# Project 1

## <Liar Dice>

## <Version 1.0>

Name: Haolan Ye (Benjamin)

Class: CSC-17A 43950

Date: 04/26/2015

# Contents

# 1. Introduction( http://en.wikipedia.org/wiki/Liar's_dice ):

Five dice are used per player with dice cups used for concealment. Each round, each player rolls a "hand" of dice under their cup and looks at their hand while keeping it concealed from the other players.

The first player begins bidding, announcing any face value and the number of dice that the player believes are showing that value, under all of the cups in the game. Ones are often wild, always counting as the face of the current bid. Each player has two choices during their turn: to make a higher bid, or challenge the previous bid - typically with a call of "liar." Raising the bid means either increasing the quantity, or the face value, or both.

If the current player challenges the previous bid, all dice are revealed. If the bid is valid (at least as many of the face value and any wild aces are showing as were bid), the bidder wins. Otherwise, the challenger wins.

That game I made was created based on the way I played in China. Because of difference between different regions, some rules of Liar Dice are different. In China, there are some addition rules (also in my game):

- From the beginning, ones are wild unless you bid 3 ones
- After bidding ones, ones cannot be wild anymore
- You could bid only 3 fives, which means you bid there are 3 fives but not including ones
- After bidding only 3 fives, ones cannot be wild anymore
- The number of face of first bid has to be greater than 1.5*players

# 2. Summary:

| Total Line of Code | 406 |
|---|---|
| Comment Line | 45 |
| Variable | 26 |
| Function | 14 |

This game contains most concepts that we have learned in the class. I used pointer with player (structure) and used structure to record the dices that each player has. In the structure of player, there is also a tag (integer) for the player. I will use the tag when someone wants to challenge. The game will write the data of players into binary file, and after someone challenge, it read the file to a new players array. Afterward, it will get the result by using the new players array.

# 3. Problems during coding

a) **Limit the player input with correct format**

When player doesn't challenge, he need to make a higher bid. Player needs to input a string for bidding. "4 5" means that player bids 4 fives. "4n5" means that player bids 4 fives only (ones cannot be wild at that time).

b) **Get the playing order for the players**

At the beginning of the game, it will randomly get the playing order for the players. In the rest of the game, players bid and challenge based on that order. I used a switch statement in a do-while statement to randomly access. It loops until someone challenges.

**c) What should AI do?**

There is no a specific algorithm for the AI in that game. I made the AI based on what I think when I play Liar Dice. There are lots of possibilities that happens when AI determine challenging or not.

- When AI doesn't have the dices that bided by previous players
- When AI only have one that bided by previous players

… etc.

When AI needs to bid higher, AI should sometimes lie and sometimes tell the truth. Therefore, I set the possibility that AI lie to 2/5. When AI tells the truth, he will bid based on what dices he has. When AI lies, he will randomly select one face of dice that does not exist in his dices.

**d) One is wild**

Mostly, ones are wild unless you bid "3 fives only" or "3 ones" (Both are example). Therefore, I need a Boolean to record one is wild or not. After one is not wild, the number of each dices doesn't count ones.

# 4.Pseudo Code

Set seed for random number
Introduce the game
Prompt players for the number of players
Roll the dices for players
Write the data of players into binary file
Initialize based on the number of player
Display player's dices
Randomly choose a player be the first bidder
Begin biding until someone challenge {
  Someone bid first (based on random select)
  Other players determine challenge or not
  Bid in players order
}
Read the binary file and save the data to a new array of structure
Show dices of all players
Display the result of the game {
    Check the number of face bided is greater than the number of dices that players have exactly
}
Deallocate memory

# 5. Screen Shot

1) Ask for number of player

```
Welcome to Liar Dice

Number of player(2(Easy) or 3(Hard)):
```

2) Randomly choose a player to be the first bid, then you can challenge

```
Welcome to Liar Dice

Number of player(2(Easy) or 3(Hard)): 3
Write to the file...

Your    dice: 6 5 5 3 5
AI #1 bid 4  4s
Would you like to challenge?(Y or N):
```

3) If challenge, the result will come up

```
Welcome to Liar Dice

Number of player(2(Easy) or 3(Hard)): 3
Write to the file...

Your    dice: 6 5 5 3 5
AI #1 bid 4  4s
Would you like to challenge?(Y or N): y
You Challenge
Read from the file...


Your    dice: 6 5 5 3 5
AI #1's dice: 4 4 4 5 5
AI #2's dice: 5 5 5 3 4

Totally, there are 4 4s
Your challenge failed
```

4) If not challenge, your turn to bid

```
Welcome to Liar Dice

Number of player(2(Easy) or 3(Hard)): 3
Write to the file...

Your    dice: 4 1 3 1 5
AI #2 bid 4  3s
Would you like to challenge?(Y or N): n
AI #1 does not challenge
Your bidding: format:"3 4"(means u bid 3 4s,and 1s are wild) or "4n5"
Your bidding:
```

5) After you bid, if AI(s) doesn't challenge, it's their turn to bid, and ask you challenge or not

```
Welcome to Liar Dice

Number of player(2(Easy) or 3(Hard)): 3
Write to the file...

Your    dice: 4 1 3 1 5
AI #2 bid 4  3s
Would you like to challenge?(Y or N): n
AI #1 does not challenge
Your bidding: format:"3 4"(means u bid 3 4s,and 1
Your bidding: 4 5
You bid 4  5s
AI #1 does not challenge
AI #2 does not challenge
AI #1 bid 4  6s
Would you like to challenge?(Y or N):
```

# 6. System Libraries

- ✓ **<iostream>**
- ✓ **<cstdlib>**
- ✓ **<ctime>**
- ✓ **<string>**
- ✓ **<vector>**
- ✓ **<fstream>**

# 7. Variables List

| Type | Variable Name | Description | Declare Location(line) |
|---|---|---|---|
| fstream | out | Output to the file | 109 |
| | in | Read from the file | 119 |
| int | numPyr | Number of player | 42 |
| | round | How many round | 43 |
| | open | A tag for open player | 44 |
| | temp | Temp for randomly get a player | 65 |
| | numTemp | Number of face bided | 180 |
| | num | Number of one face of dice | 310 |
| | ones | How many ones in dices | 311 |
| | cnt | Count for one face of dice | 366 |
| | hgst | Highest frequency in dices | 367 |
| | indx | Index of highest frequency | 368 |
| | total | Total number of one faces in three players' hand | 392 |
| int * | temp | Record the frequency | 365 |
| string | input | Let player to input then check | 46 |

| | | | |
|---|---|---|---|
| | ans | Answer of challenge or not | 161 |
| | bid | What player bid | 179 |
| Player * | players | Pointer for players | 57 |
| | copy | Pointer for players from file | 58 |
| char | fceTemp | Face of dice bided | 181 |
| char * | dices | Dices for one player | 141 |
| char | guess | The number player guess | 62 |
| vector<char> | nExist | Faces not exist in dices | 322 |
| | exist | Face exist in dices | 351 |
| boolean | wild | Ones are wild or not | 45 |
| | invalid | Input invalid or not | 182 |

# 8. Function List

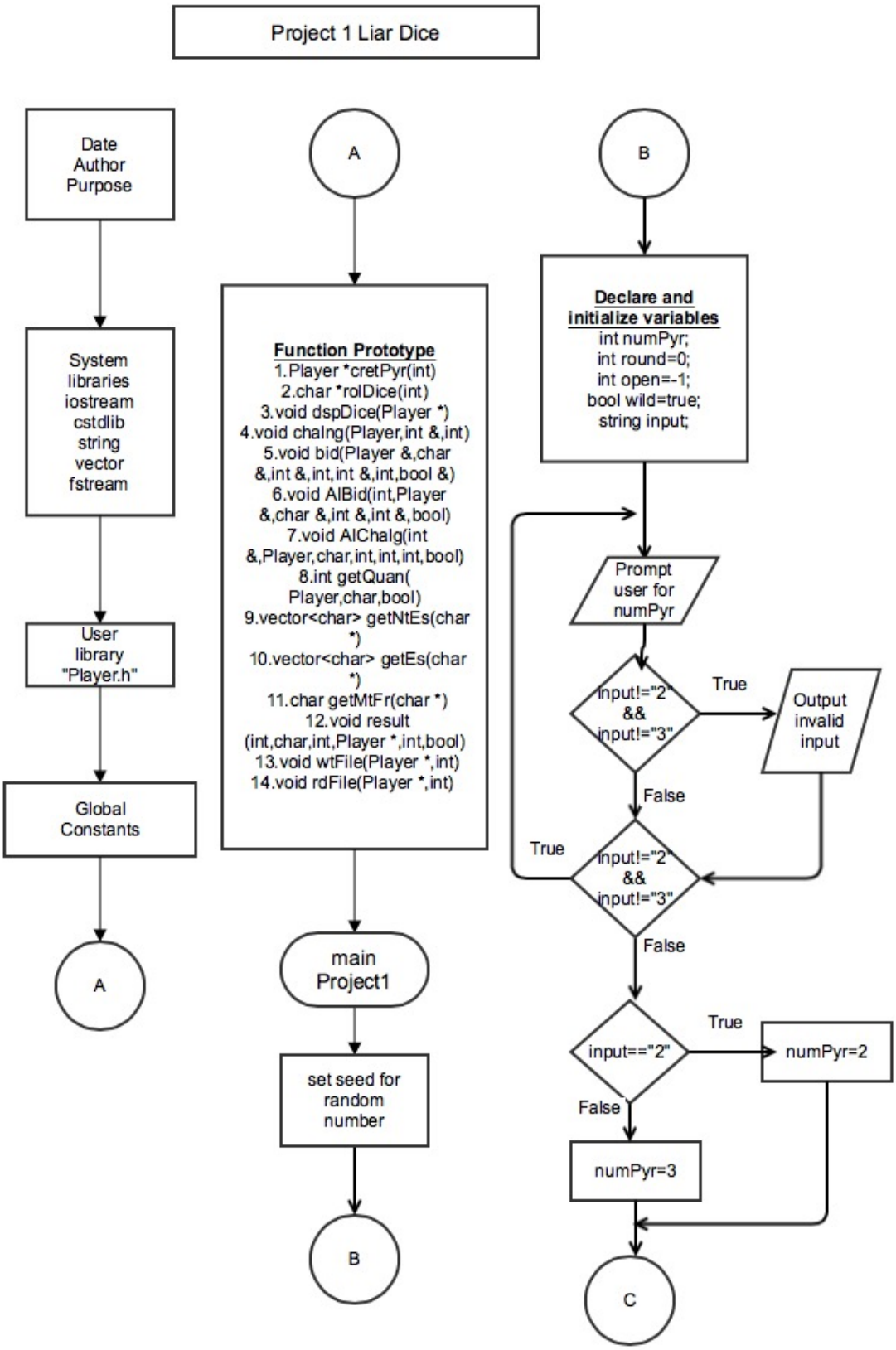
| Type | Name | Argument | Function | Location |
|---|---|---|---|---|
| Player * | cretPyr | int | create player and roll dice | 129 |
| char * | rolDice | int | roll 5 dices | 139 |
| void | dspDice | Player * | display dice of a player | 150 |
| void | chalng | Player,int &,int | Player challenge | 160 |
| void | bid | Player &,char &,int &,int,int &,int,bool & | Player bid | 178 |
| void | AIBid | int,Player &,char &,int &,int &,bool | AI bid | 266 |
| void | AIChalg | int&,Player,char,int,int,int,bool | AI challenge or not | 237 |
| int | getQuan | Player,char,bool | Get the quantity of that face of dice in one AI's hand | 309 |
| vector< char> | getNtEs | char * | Get the dices that not exist one AI's hand | 321 |
| vector< char> | getEs | char * | Get the dices that exist one AI's hand | 349 |
| char | getMtFr | char * | get the most frequent face of dices in one AI's hand | 364 |
| void | result | int,char,int,Player *,int,bool | Determine who win and lost | 391 |
| void | wtFile | Player *,int | write the array of Player into file | 108 |
| void | rdFile | Player *,int | Read the file | 118 |

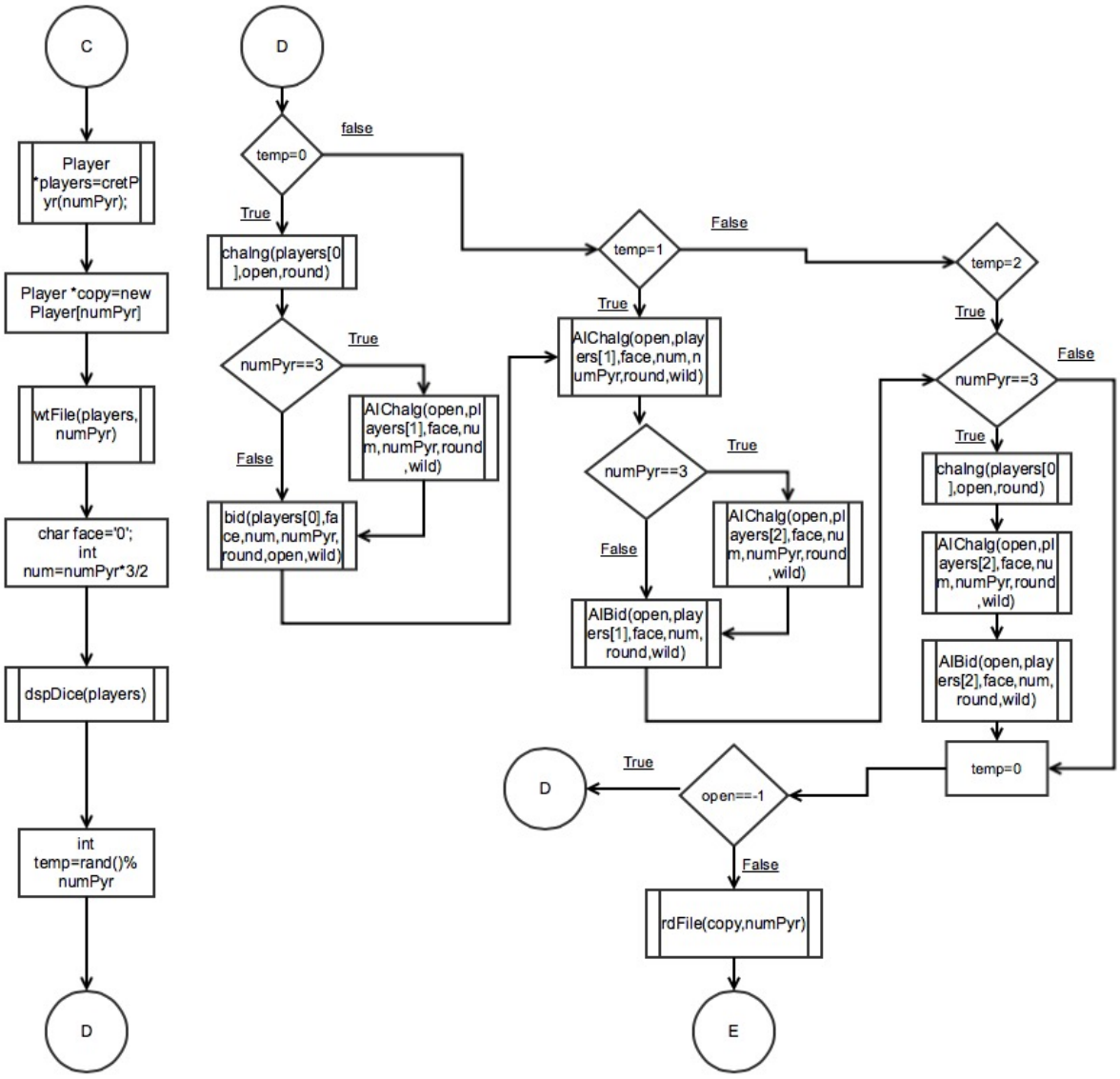
# 9. Concept covered

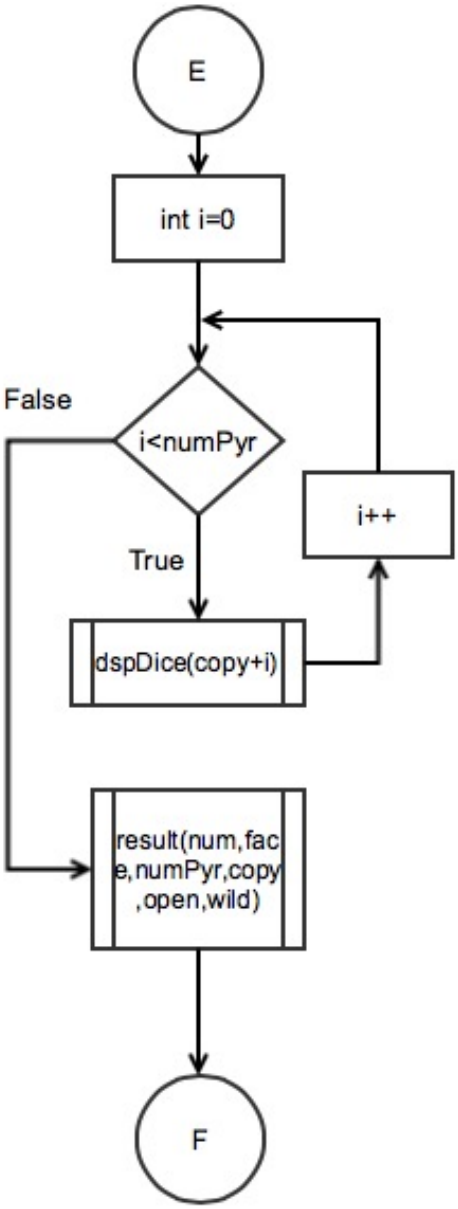
| Concept | Type | Code | Location(line) |
|---|---|---|---|
| Pointer with structure | Player * | Player *players=cretPyr(numPyr); | 57 |
| Function return pointer | | Player *cretPyr(int); | 23 |
| Functionwith structure | | Player *cretPyr(int); | 23 |
| Point with array | int * | int *temp=new int[5]; | 365 |
| Type casting | static_cast<type> | static_cast<unsigned short>(time(0)) | 41 |
| Binary file | fstream output | fstream out; | 109 |
| | fstream input | fstream in; | 119 |
| string | string | string ans="N" | 161 |
| Switch | switch | switch(temp) | 68 |
| Loop | for | for(int i=1;i<=6;i++) | 324 |
| | do-while | do {} while(open==-1) | 67 |
| Function | void, int, string, char, bool | string toDash(int) | 23 |

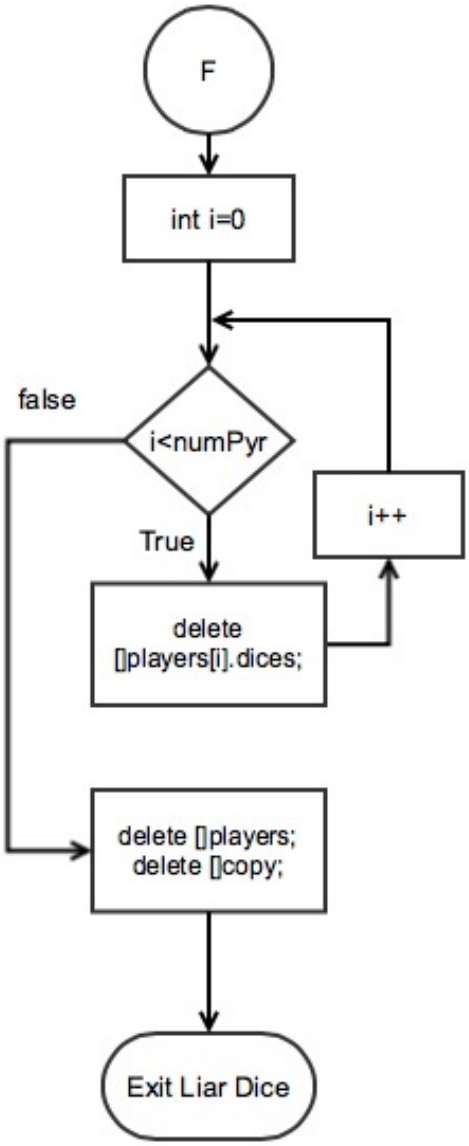
| | | | |
|---|---|---|---|
| vector | vector<char> | vector<char> nExist | 322 |
| Sorting | Sorting | for(int i=0;i<5;i++) {<br>for(int j=i+1;j<6;j++) {<br>if(nExist[i]<nExist[j]) {<br>char temp=nExist[i];<br>nExist[i]=nExist[j];<br>nExist[j]=temp;<br>}<br>}<br>} | 332 |
| Random number | char | srand(static_cast<unsigned short>(time(0)));<br>int temp=rand()%numPyr; | 65 |

# 10. Flowchart (1) Main flowchart (3 pages)

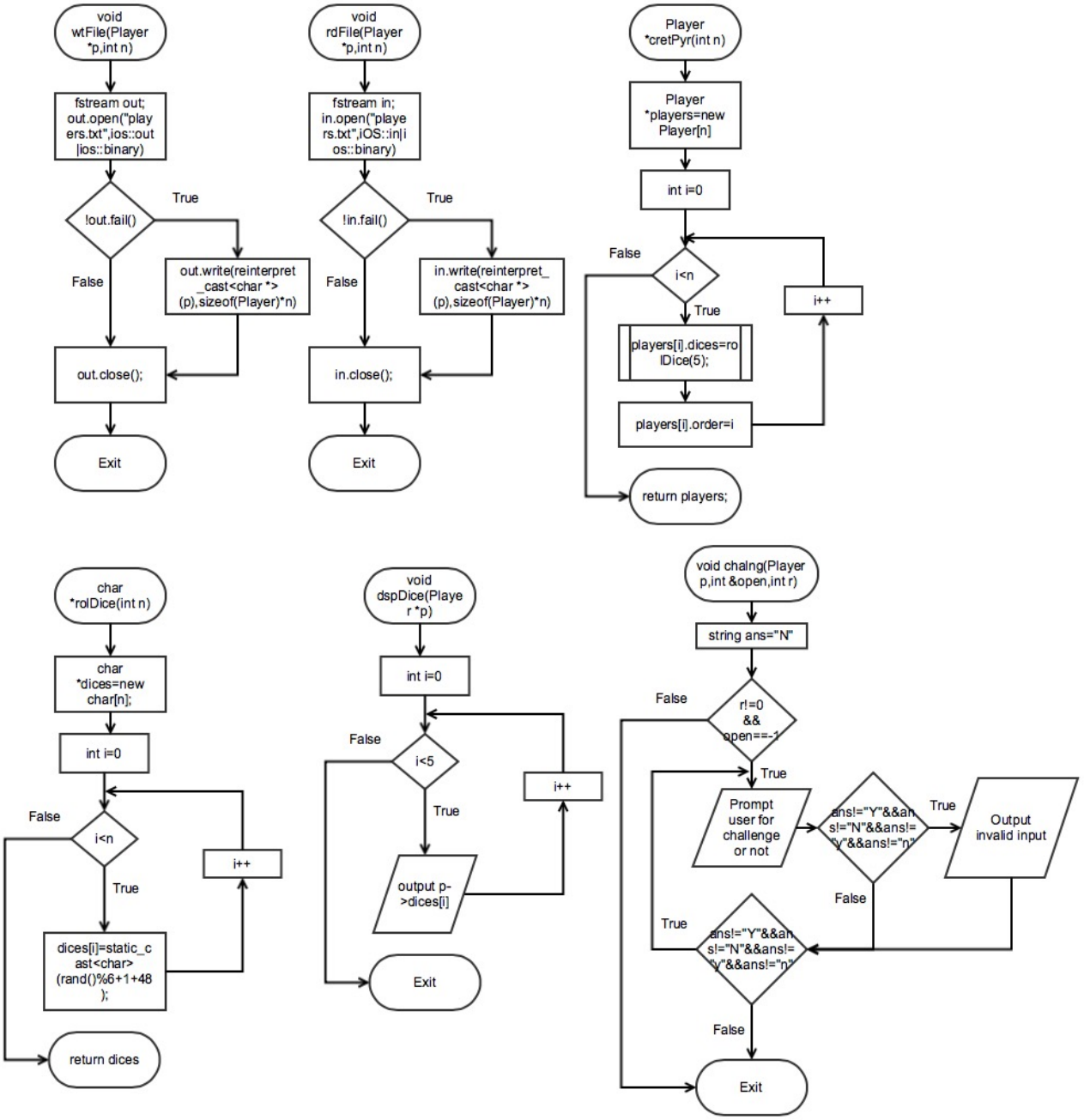


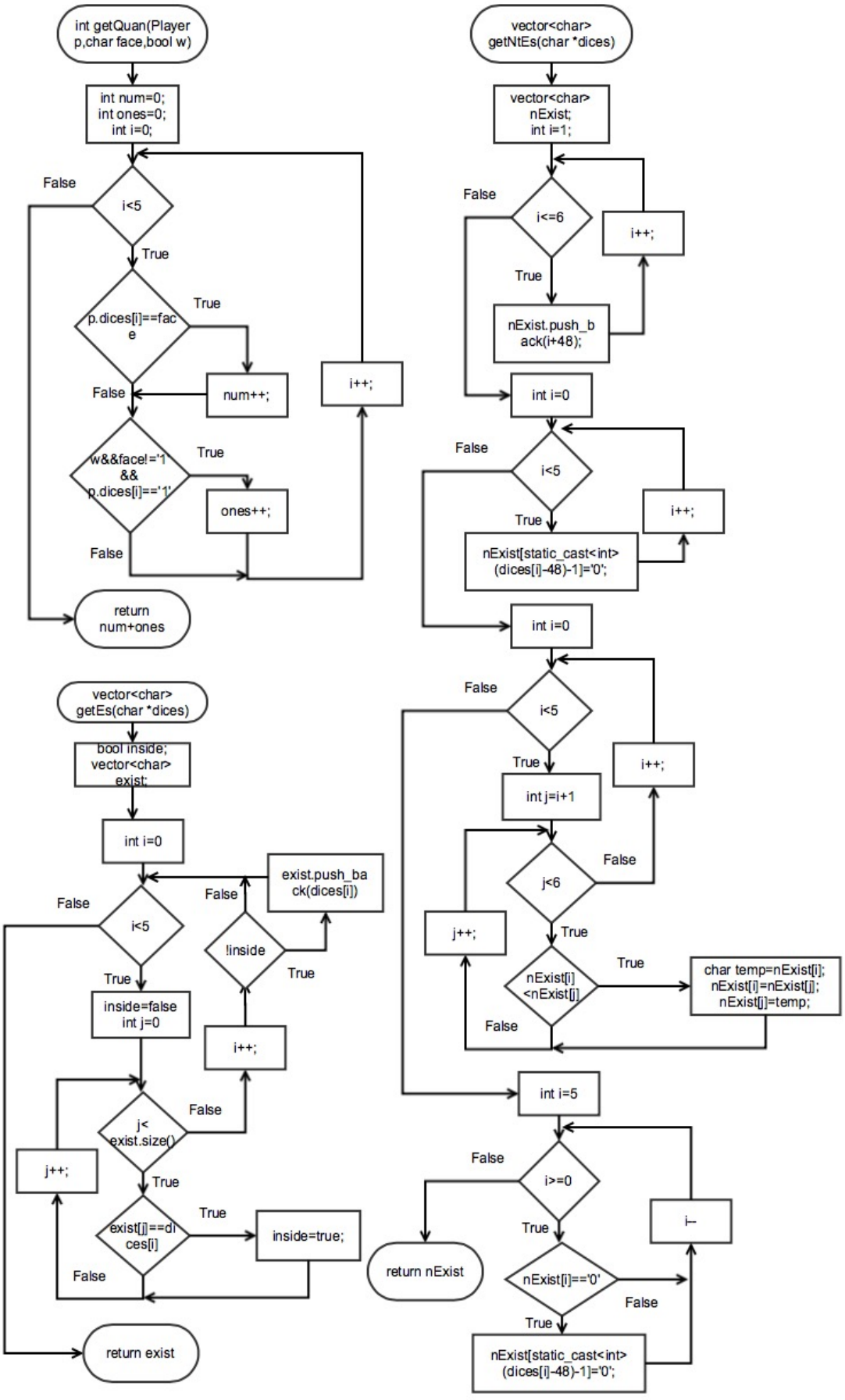

C
Player *players=cretPyr(numPyr);
Player *copy=new Player[numPyr]
wtFile(players, numPyr)
char face='0'; int num=numPyr*3/2
dspDice(players)
int temp=rand()% numPyr
D
D
temp=0
false
True
chalng(players[0],open,round)
numPyr==3
True
AIChalg(open,players[1],face,num,numPyr,round,wild)
False
bid(players[0],face,num,numPyr,round,open,wild)
temp=1
False
True
AIChalg(open,players[1],face,num,numPyr,round,wild)
numPyr==3
True
AIChalg(open,players[2],face,num,numPyr,round,wild)
False
AIBid(open,players[1],face,num,round,wild)
temp=2
True
numPyr==3
False
True
chalng(players[0],open,round)
AIChalg(open,players[2],face,num,numPyr,round,wild)
AIBid(open,players[2],face,num,round,wild)
temp=0
D
True
open==-1
False
rdFile(copy,numPyr)
E

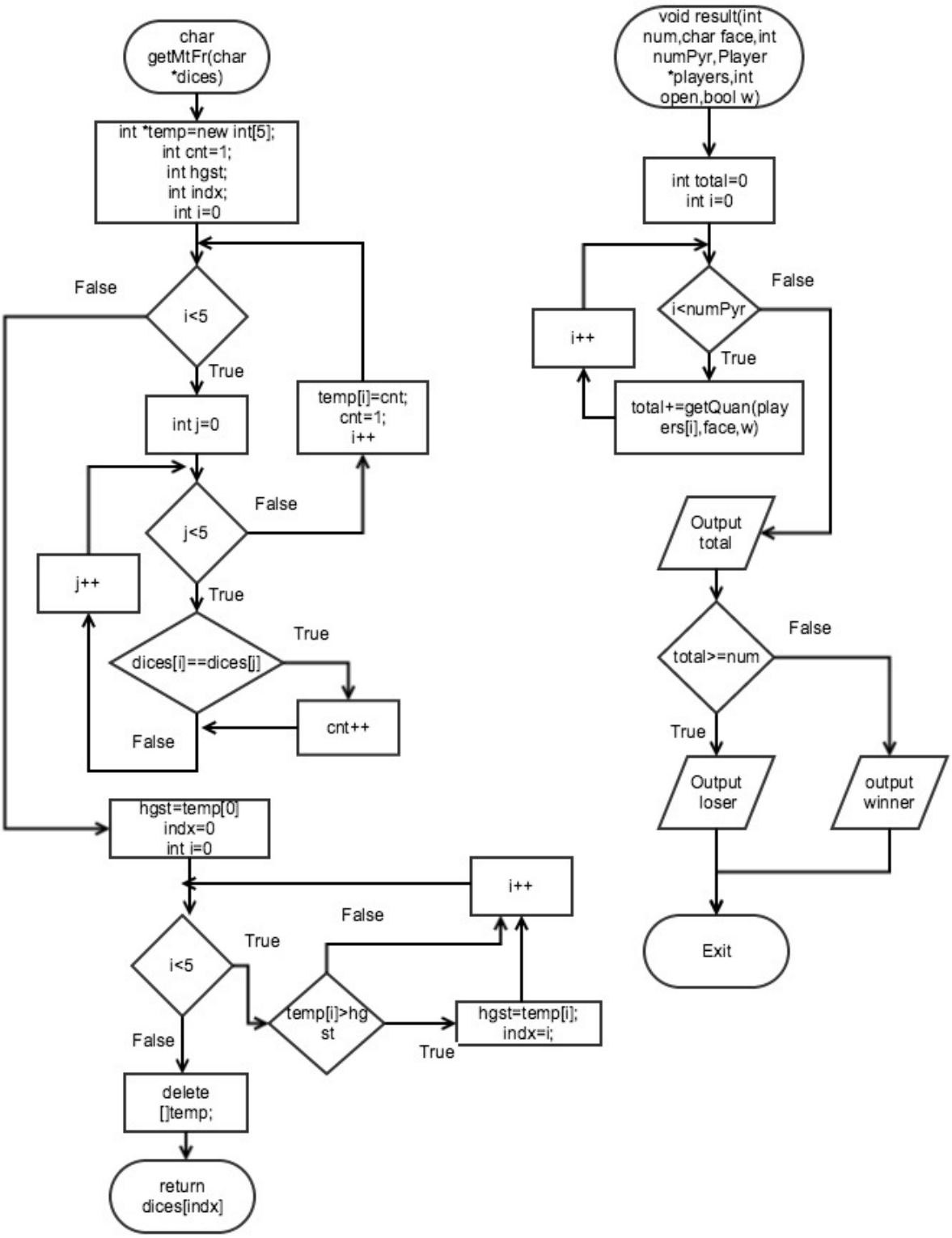

E
int i=0
False
i<numPyr
True
i++
dspDice(copy+i)
result(num,face,numPyr,copy,open,wild)
F

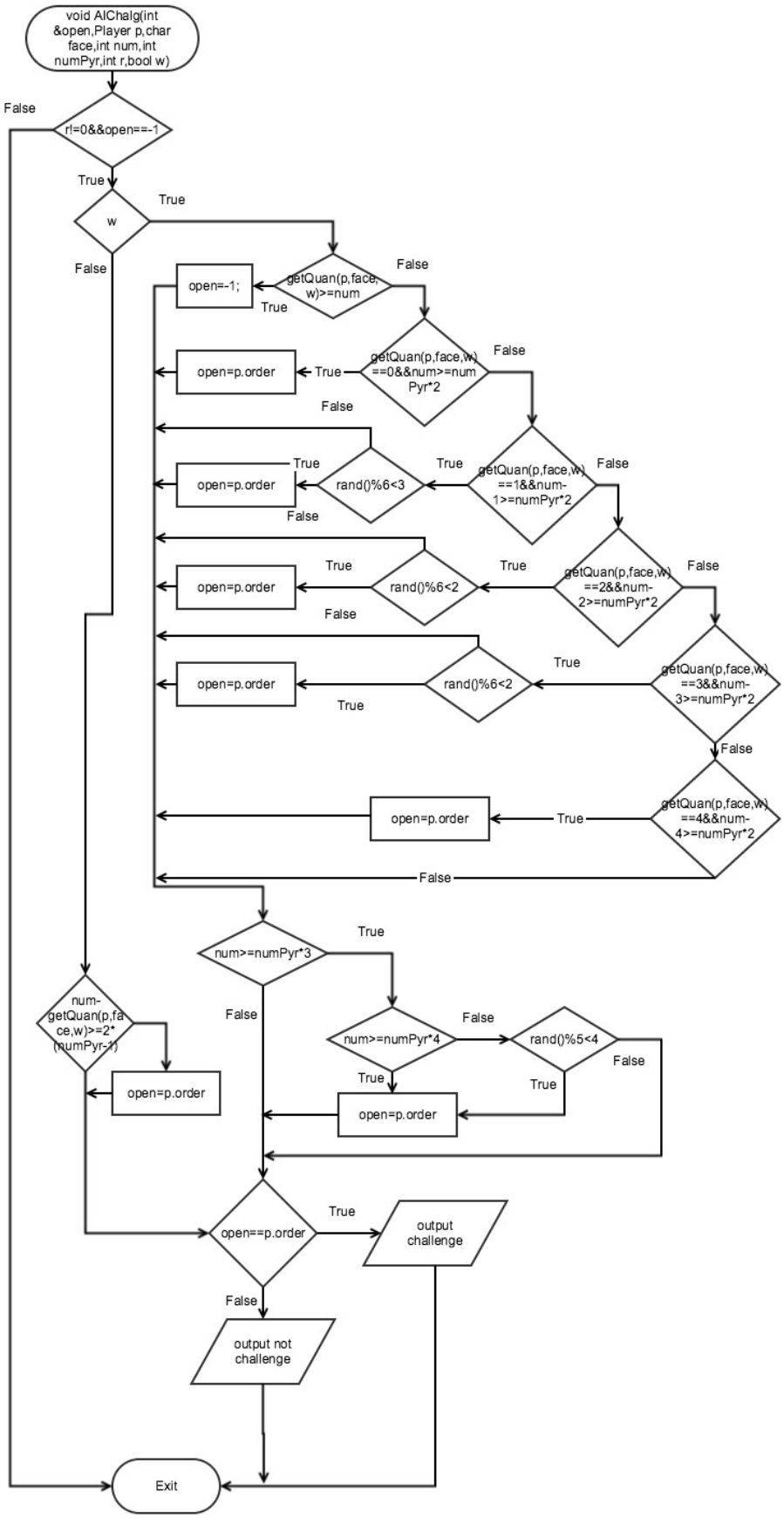

F
int i=0
false
i<numPyr
True
i++
delete []players[i].dices;
delete []players;
delete []copy;
Exit Liar Dice

# (2) Function flowchart (3 pages)




int getQuan(Player p,char face,bool w)
int num=0;
int ones=0;
int i=0;
i<5
False
True
p.dices[i]==face
True
num++;
False
w&&face!='1' && p.dices[i]=='1'
True
ones++;
False
i++;
return num+ones
vector<char> getNtEs(char *dices)
vector<char> nExist;
int i=1;
i<=6
False
True
nExist.push_back(i+48);
i++;
int i=0
i<5
False
True
nExist[static_cast<int>(dices[i]-48)-1]='0';
i++;
int i=0
i<5
False
True
int j=i+1
j<6
False
True
nExist[i]<nExist[j]
True
char temp=nExist[i];
nExist[i]=nExist[j];
nExist[j]=temp;
False
j++;
i++;
int i=5
i>=0
False
True
nExist[i]=='0'
True
False
nExist[static_cast<int>(dices[i]-48)-1]='0';
i--
return nExist
vector<char> getEs(char *dices)
bool inside;
vector<char> exist;
int i=0
i<5
False
True
inside=false
int j=0
j<exist.size()
False
True
exist[j]==dices[i]
True
inside=true;
False
j++;
!inside
True
False
exist.push_back(dices[i])
i++;
return exist


char getMtFr(char *dices)
int *temp=new int[5];
int cnt=1;
int hgst;
int indx;
int i=0
i<5
False
True
int j=0
temp[i]=cnt;
cnt=1;
i++
j<5
False
True
j++
dices[i]==dices[j]
True
False
cnt++
hgst=temp[0]
indx=0
int i=0
i++
i<5
True
False
temp[i]>hgst
True
hgst=temp[i];
indx=i;
False
delete []temp;
return dices[indx]
void result(int num,char face,int numPyr,Player *players,int open,bool w)
int total=0
int i=0
i<numPyr
False
True
i++
total+=getQuan(players[i],face,w)
Output total
total>=num
False
True
Output loser
output winner
Exit


void AIChalg(int &open,Player p,char face,int num,int numPyr,int r,bool w)
r!=0&&open==-1
False
True
w
True
False
getQuan(p,face,w)>=num
False
True
open=-1;
getQuan(p,face,w)==0&&num>=numPyr*2
True
open=p.order
False
getQuan(p,face,w)==1&&num-1>=numPyr*2
True
rand()%6<3
True
open=p.order
False
False
getQuan(p,face,w)==2&&num-2>=numPyr*2
True
rand()%6<2
True
open=p.order
False
False
getQuan(p,face,w)==3&&num-3>=numPyr*2
True
rand()%6<2
True
open=p.order
True
False
getQuan(p,face,w)==4&&num-4>=numPyr*2
True
open=p.order
False
num>=numPyr*3
True
False
num-getQuan(p,face,w)>=2*(numPyr-1)
open=p.order
num>=numPyr*4
False
rand()%5<4
False
True
True
open=p.order
open==p.order
True
output challenge
False
output not challenge
Exit

# 11. Code

```
/*
 * File:   main.cpp
 * Author: Haolan Ye(Benjamin)
 * Created on April 21, 2015, 10:29 AM
 * Purpose: CSC-17A Project 1 Liar Dice
 */

//System libraries
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
using namespace std;

//User libraries
#include "Player.h"

//Global Constants

//Function prototypes
Player *cretPyr(int);//create player and roll dice
char *rolDice(int);//roll 5 dices
void dspDice(Player *);//display dice of a player
void chalng(Player,int &,int);//Player challenge
void bid(Player &,char &,int &,int,int &,int,bool &);//First player(you)
void AIBid(int,Player &,char &,int &,int &,bool);//AI's turn
void AIChalg(int &,Player,char,int,int,int,bool);//AI challenge
int getQuan(Player,char,bool);//get the quantity of that face of dice in one AI's hand
vector<char> getNtEs(char *);//get the dices that not exist one AI's hand
vector<char> getEs(char *);//get the dices that exist one AI's hand
char getMtFr(char *);//get the most frequent face of dices in one AI's hand
void result(int,char,int,Player *,int,bool);//Determine who win and lost
void wtFile(Player *,int);//write the array of Player into file
void rdFile(Player *,int);//read the file

//Execution begins here
int main(int argc, char** argv) {
    //set seed for rolling dice
    srand(static_cast<unsigned int>(time(0)));
    int numPyr;//number of player
    int round=0;//round of the game
    int open=-1;//open=-1 -> not open; =0 -> player1 open; =1 -> player2 open etc.
    bool wild=true;//1 is wild, when after bidding 1s or bidding only 3 5s,then 1 is not wild
    string input;//temp for input
```

```
    cout<<"Welcome to Liar Dice"<<endl<<endl;
    //Prompt user for number of player
    do {
       cout<<"Number of player(2(Easy) or 3(Hard)): ";
       cin>>input;
       if(input!="2"&&input!="3")
          cout<<"Invalid input"<<endl;
    } while(input!="2"&&input!="3");
    numPyr=(input=="2"?2:3);
    //create players and roll dices
    Player *players=cretPyr(numPyr);
    Player *copy=new Player[numPyr];//create for reading file
    wtFile(players,numPyr);//write all the players into file
    //Initialize based on the number of player
    char face='0';//initial the face to 0
    int num=numPyr*3/2;//initial the number to 1.5*number of player
    dspDice(players);//display your dice
    //game begins
    int temp=rand()%numPyr;//randomly select who is the first to bid
    //run until somebody challenges
    do {
       switch(temp) {
          case 0: {
             chalng(players[0],open,round);//player challenge
             if(numPyr==3) AIChalg(open,players[1],face,num,numPyr,round,wild);//AI #2
challenge
             bid(players[0],face,num,numPyr,round,open,wild);//player bid
          }
          case 1: {
             AIChalg(open,players[1],face,num,numPyr,round,wild);//AI #1 challenge
             if(numPyr==3) AIChalg(open,players[2],face,num,numPyr,round,wild);//AI #2
challenge
             AIBid(open,players[1],face,num,round,wild);//AI #1 bid
          }
          case 2: {
             if(numPyr==3) {
                chalng(players[0],open,round);//Player challenge
                AIChalg(open,players[2],face,num,numPyr,round,wild);//AI #2 challenge
                AIBid(open,players[2],face,num,round,wild);//AI #2 bid
             }
          }
       }
       temp=0;
    } while(open==-1);
    //read the binary file(players)
    rdFile(copy,numPyr);
    //show dices of all players
    for(int i=0;i<numPyr;i++) {
       dspDice(copy+i);
    }
```

```
    //display the result of the game
    result(num,face,numPyr,copy,open,wild);
    //deallocate memory
    for(int i=0;i<numPyr;i++) {
        //delete []copy[i].dices;
        delete []players[i].dices;
    }
    delete []players;
    delete []copy;
    //Exit stage right
    return 0;
}

void wtFile(Player *p,int n) {
    fstream out;
    cout<<"Write to the file..."<<endl;
    out.open("players.txt",ios::out|ios::binary);
    if(!out.fail()) {
      out.write(reinterpret_cast<char *>(p),sizeof(Player)*n);
    }
    out.close();
}

void rdFile(Player *c,int n) {
    fstream in;
    cout<<"Read from the file..."<<endl<<endl;
    in.open("players.txt",ios::in|ios::binary);
    if(!in.fail()) {
      in.read(reinterpret_cast<char *>(c),sizeof(Player)*n);
    }
    in.close();
}

//create players and roll dices
Player *cretPyr(int n) {
    Player *players=new Player[n];
    for(int i=0;i<n;i++) {
        players[i].dices=rolDice(5);
        players[i].order=i;
    }
    return players;
}

//roll dices and save in char array
char *rolDice(int n) {
    //allocate memory
    char *dices=new char[n];
    //randomly roll the dice
    for(int i=0;i<n;i++) {
        dices[i]=static_cast<char>(rand()%6+1+48);
```

```
    }
    return dices;
}

//output the dices of a player
void dspDice(Player *p) {
    if(p->order==0) cout<<endl<<"Your    ";
    else cout<<"AI #"<<p->order<<"'s ";
    cout<<"dice: ";
    for(int i=0;i<5;i++) {
        cout<<p->dices[i]<<" ";
    }
    cout<<endl;
}

void chalng(Player p,int &open,int r) {
    string ans="N";//answer of open or not
    //prompt user for challenge or not
    if(r!=0&&open==-1) {
        do {
            cout<<"Would you like to challenge?(Y or N): ";
            cin>>ans;
            if(ans!="Y"&&ans!="N"&&ans!="y"&&ans!="n")
                cout<<"Invalid input"<<endl;
        } while(ans!="Y"&&ans!="N"&&ans!="y"&&ans!="n");
    }
    //when answer is open
    if(ans=="Y"||ans=="y") {
        open=p.order; //set open to true
        cout<<"You Challenge"<<endl;
    }
}

void bid(Player &p,char &face,int &num,int numPyr,int &r,int open,bool &w) {
    string bid;
    int numTemp;
    char fceTemp;
    bool invalid;
    //when answer is open
    if(open==-1) { //when answer is not open
        cin.ignore();
        do {
            numTemp=0;
            fceTemp=' ';
            invalid=false;
            if(r<=2) {
            cout<<"Your bidding: ";
            cout<<"format:\"3 4\"(means u bid 3 4s,and 1s are wild) or \"4n5\"(means you bid 4 5s
only, and 1s are not wild)"<<endl;
            cout<<"First bid must be >= 1.5*players"<<endl;
```

```
        }
        cout<<"Your bidding: ";
        getline(cin,bid);//1st element is number of dice,2nd is space or n,3rd is face of dice
        //check the input valid or not
        if(bid.length()!=3&&bid.length()!=4) invalid=true;//length only 3 or 4
        if(bid.length()==3||bid.length()==4) {
           for(int i=0;i<bid.length();i++) {
              if(i==bid.length()-2) {
                 if(bid.at(i)!=' '&&bid.at(i)!='n'&&bid.at(i)!='N') invalid=true;
              }
              if(i<bid.length()-2) //number of one face of dice should be a integer
                 if(bid.at(i)<'0'||bid.at(i)>'9') invalid=true;
              if(i>bid.length()-2) //face of dice should be between 1 and 6
                 if(bid.at(i)<'1'||bid.at(i)>'6') invalid=true;
           }
        }
        if(!invalid) {
           if(bid.length()==3) {
              numTemp=static_cast<int>(bid.at(0)-48);
              fceTemp=bid.at(2);
           } else if(bid.length()==4) {
              numTemp=static_cast<int>(bid.at(0)-48)*10+static_cast<int>(bid.at(1)-48);
              fceTemp=bid.at(3);
           }
        }
        //if format of input is right, check the contents of input
        if(numTemp<num) invalid=true; //quantity less than previous one
        //quantity=previous one,but face of dice< previous one
        if(numTemp==num&&fceTemp<=face) invalid=true;
        if(numTemp>numPyr*5) invalid=true;
        if(invalid) cout<<"Invalid input!!"<<endl;
     } while(invalid);

     num=numTemp;
     face=fceTemp;
     if(bid.at(bid.length()-2)=='n'||bid.at(bid.length()-2)=='N') w=false;
     if(bid.at(bid.length()-1)=='1') w=false;
     cout<<"You bid "<<num<<"  "<<face<<"s";
     if(w) cout<<" "<<endl;
     else cout<<" only"<<endl;
     r++;
  }
}

void AIChalg(int &open,Player p,char face,int num,int numPyr,int r,bool w) {
  if(r!=0&&open==-1) {
     //determine challenge or not
     if(w) {
        if(getQuan(p,face,w)>=num) open=-1; //when bided number of a kind dice <= AI's, not
challenge
```

```
            else if(getQuan(p,face,w)==0&&num>=numPyr*2) open=p.order;
            else if(getQuan(p,face,w)==1&&num-1>(numPyr-1)*2) {
               if(rand()%6<3) open=p.order; //50% to open
            } else if(getQuan(p,face,w)==2&&num-2>(numPyr-1)*2) {
               if(rand()%6<2) open=p.order; //1/3 to open
            } else if(getQuan(p,face,w)==3&&num-3>(numPyr-1)*2) {
               if(rand()%6<2) open=p.order; //1/3 to open
            } else if(getQuan(p,face,w)>=4&&num-getQuan(p,face,w)>(numPyr-1)*2) {
               open=p.order; // 100% to open
            }
            if(num>=numPyr*3) {
               if(num>=numPyr*4) open=p.order;
               else {
                  if(rand()%5<4) open=p.order;
               }
            }
         } else {
            if(num-getQuan(p,face,w)>=2*(numPyr-1)) open=p.order;
         }
         if(open==p.order) cout<<"AI #"<<p.order<<" challenge"<<endl;
         else cout<<"AI #"<<p.order<<" does not challenge"<<endl;
      }
   }

   void AIBid(int open,Player &p,char &face,int &num,int &r,bool w) {
      //bid
      if(open==-1) {
         char faceTem;
         if(w) {
            vector<char> nExist=getNtEs(p.dices);
            //truth 3/5
            if(rand()%5>=2||(nExist.size()==1&&nExist[0]=='1')) {
               if(rand()%3<2&&face==getMtFr(p.dices))  { //get the most frequent face of of AI's
   dices
                  faceTem=face;
               } else { //randomly get a dice from existed dices
                  vector<char> exist=getEs(p.dices);

                  do {
                     faceTem=exist[rand()%exist.size()];
                  } while(faceTem=='1');
               }

               if(faceTem<=face) num++;
               face=faceTem;


            } else { //lie 2/5
               //get the face of dice that AI doesn't have
               char faceTem;
```

```
            do {
               faceTem=nExist[rand()%nExist.size()];
            } while(faceTem=='1');
            if(faceTem<=face) num++;
            face=faceTem;
         }
      } else {
         face=getMtFr(p.dices);
         if(faceTem<=face) num++;
      }
      cout<<"AI #"<<p.order<<" bid "<<num<<"  "<<face<<"s";
      if(w) cout<<" "<<endl;
      else cout<<" only"<<endl;
      r++;
   }
}

//get the quantity of one face of dice of one player
int getQuan(Player p,char face,bool w) {
   int num=0;
   int ones=0;
   for(int i=0;i<5;i++) {
      if(p.dices[i]==face) num++;
      if(w&&face!='1'&&p.dices[i]=='1') ones++;
      //when 1 is not wild
   }
   return num+ones;
}

//get the faces of dice that doesn't exist in AI's hand
vector<char> getNtEs(char *dices) {
   vector<char> nExist;//not exist face of dice
   //initialize 6 elements from 1 to 6
   for(int i=1;i<=6;i++) {
      nExist.push_back(i+48);
   }
   //when the face of the dices comes up, set that face in the vector to 0
   for(int i=0;i<5;i++) {
      nExist[static_cast<int>(dices[i]-48)-1]='0';
   }
   //sort the vector form high to low
   for(int i=0;i<5;i++) {
      for(int j=i+1;j<6;j++) {
         if(nExist[i]<nExist[j]) {
            char temp=nExist[i];
            nExist[i]=nExist[j];
            nExist[j]=temp;
         }
      }
   }
```

```
    //take out the existing number
    for(int i=5;i>=0;i--) {
        if(nExist[i]=='0') nExist.pop_back();
    }
    return nExist;//return the vector
}

//get the faces of dice that exist in AI's hand
vector<char> getEs(char *dices) {
    bool inside;
    vector<char> exist;
    for(int i=0;i<5;i++) {
        inside=false;
        //use for loop to get the existing dices
        for(int j=0;j<exist.size();j++) {
            if(exist[j]==dices[i]) inside=true;
        }
        if(!inside) exist.push_back(dices[i]);
    }
    return exist;
}

//get the most frequent face of dice in the dices
char getMtFr(char *dices) {
    int *temp=new int[5];
    int cnt=1;//count for the dice
    int hgst;//highest number
    int indx;//index
    //if dices: 2 2 3 4 2, then temp: 3 3 1 1 3
    for(int i=0;i<5;i++) {
        for(int j=0;j<5;j++) {
            if(dices[i]==dices[j]) cnt++;
        }
        temp[i]=cnt;
        cnt=1;
    }
    hgst=temp[0];//initialize the highest number
    indx=0;//initialize the index
    //find out the highest and its index
    for(int i=0;i<5;i++) {
        if(temp[i]>hgst) {
            hgst=temp[i];
            indx=i;
        }
    }
    delete []temp;//delete allocate memory
    return dices[indx];
}

//print out the result
```

```
void result(int num,char face,int numPyr,Player *players,int open,bool w) {
   int total=0;
   //count the face of all players
   for(int i=0;i<numPyr;i++) {
      total+=getQuan(players[i],face,w);
   }
   cout<<endl<<"Totally, there are "<<total<<" "<<face<<"s"<<endl;
   if(total>=num) {
      if(open==0) cout<<"Your challenge failed"<<endl;
      else cout<<"AI #"<<open<<"'s challenge failed"<<endl;
   } else {
      if(open==0) cout<<"Your challenge succeed"<<endl;
      else cout<<"AI #"<<open<<"'s challenge succeed"<<endl;
   }
}
```