```c
#include "stdafx.h"
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_NAME 20
#define STAGES 15
typedef struct _PLAYER
{
        char    name[MAX_NAME];
        int             sets;
}PLAYER;
typedef struct _BTREE_NODE
{
  void * data;
  struct _BTREE_NODE * left;
  struct _BTREE_NODE * right;
} BTREE_NODE;
typedef BTREE_NODE * BTREE;
typedef enum _BTREE_LOCATION {BTREE_LEFT,BTREE_RIGHT} BTREE_LOCATION;
typedef enum _BOOLEAN {FALSE = 0,TRUE = 1} BOOLEAN;
typedef enum _STATUS {ERROR = 0,OK = 1} STATUS;
#define DATA(node)      ((node)->data)
#define LEFT(node)      ((node)->left)
#define RIGHT(node)     ((node)->right)
#define EMPTY           NULL
#define NO_LINK         NULL

BTREE_NODE *    NewBtreeNode(void * data);
BTREE_NODE *    AddBtreeNode(BTREE_NODE * upnode, BTREE_NODE * node, BTREE_LOCATION where);
int                             BtreeSize(BTREE btree);
int                             BtreeDeep(BTREE btree);
BOOLEAN                 BtreeLeaf(BTREE_NODE * node);
BTREE_NODE *    InitNode(void *, BTREE_NODE *, BTREE_NODE *);
BTREE_NODE *    CreateBtree(void **, int, int);
STATUS                  ReadPlayersFromFile(void **, char *);
void                    PrintLeafs(BTREE);
void                    BtreeFree(BTREE);
void                    PrintWinnerGames(BTREE);
int                             CountTotalSets(BTREE);
int                             CountWinnerSets(BTREE, void *);
void                    PrintAllGames(BTREE);

int _tmain(int argc, _TCHAR* argv[])
{
        BTREE   Btree;
        void *  players[STAGES];
        char    file_name[MAX_NAME];

        printf("Nome do ficheiro: ");
        scanf("%s", file_name);
        if(ReadPlayersFromFile(players,file_name))
        {
                Btree = CreateBtree(players,0,STAGES);

                printf("\nLista de participantes:\n");
                PrintLeafs(Btree);

                printf("\nLista de Jogos:\n");
                PrintAllGames(Btree);

                printf("\nNúmero de eliminatórias: %d",BtreeDeep(Btree)-1);
                printf("\nNúmero de Jogos: %d",BtreeSize(Btree)/2);
                printf("\nNúmero de Sets: %d",CountTotalSets(Btree));
                printf("\nVencedor do torneio: %s\n",((PLAYER *)DATA(Btree))->name);
                printf("\nJogos disputados pelo Vencedor:\n");
                PrintWinnerGames(Btree);
                printf("\nSets ganhos pelo Vencedor: %d\n",CountWinnerSets(Btree,DATA(Btree)));
                BtreeFree(Btree);
                getch();
        }
        else
                printf("ERRO na leitura do ficheiro\n");
        getch();
        return 0;
}


BTREE_NODE * NewBtreeNode(void * data)
{
  BTREE_NODE * tmp_pt;
  if ((tmp_pt = (BTREE_NODE *)malloc(sizeof(BTREE_NODE)))!=NULL)
   {
    DATA(tmp_pt) = data;
    LEFT(tmp_pt) = RIGHT(tmp_pt) = NULL;
   }
  return tmp_pt;
}

BTREE_NODE * AddBtreeNode(BTREE_NODE * upnode, BTREE_NODE * node, BTREE_LOCATION where)
{
  BTREE_NODE * tmp_pt = upnode;

  if (where == BTREE_LEFT)
   {
    if (LEFT(upnode) == NULL)
      LEFT(upnode) = node;
     else
      tmp_pt = NULL;
   }
   else
   {
    if (RIGHT(upnode) == NULL)
      RIGHT(upnode) = node;
     else
      tmp_pt = NULL;
   }
  return tmp_pt;
}

BTREE_NODE *InitNode(void * ptr_data,BTREE_NODE * node1,BTREE_NODE * node2)
{
        BTREE_NODE * tmp_pt = NULL;

        tmp_pt = NewBtreeNode(ptr_data);
        LEFT(tmp_pt) = node1;
        RIGHT(tmp_pt) = node2;
        return(tmp_pt);
}

BTREE_NODE *CreateBtree(void ** v, int i, int size)
{
    if( i >= size)
                return(NULL);
        else
                return(InitNode(*(v+i),CreateBtree(v,2*i+1,size),CreateBtree(v,2*i+2,size)));
}

void BtreeFree(BTREE btree)
{
        if (btree != NULL)
        {
                BtreeFree(LEFT(btree));
                BtreeFree(RIGHT(btree));
                free(btree);
        }
}
```

```
int BtreeSize(BTREE btree)
{
  int count=0;

  if (btree != NULL)
        count = 1 + BtreeSize(LEFT(btree)) + BtreeSize(RIGHT(btree));
  return(count);
}

BOOLEAN BtreeLeaf(BTREE_NODE * btree)
{

  if ((LEFT(btree) == NULL) && (RIGHT(btree) == NULL))
    return(TRUE);
   else
    return(FALSE);
}

int BtreeDeep(BTREE btree)
{
  int deep=0, left, right;

  if (btree != NULL)
   {
    left = BtreeDeep(LEFT(btree));
    right = BtreeDeep(RIGHT(btree));
    deep = 1 + ((left > right)? left : right);
   }
  return(deep);
}

STATUS ReadPlayersFromFile(void ** players, char * file_name)
{
        FILE *  fp;
        int             j, i = 0;
        void *  ptr_data;

        if((fp = fopen(file_name,"r")) != NULL)
        {
                while(!feof(fp))
                {
                        if((ptr_data = malloc(sizeof(PLAYER))) != NULL)
                        {
                                fscanf(fp,"%[^;];",((PLAYER *)ptr_data)->name);
                                fscanf(fp,"%d\n",&(((PLAYER *)ptr_data)->sets));
                                players[i] = ptr_data;
                                i++;
                        }
                        else
                        {
                                for(j=i;j>=0;j--)
                                        free(players[j]);
                                return(ERROR);
                        }
                }
                fclose(fp);
                return(OK);
        }
        else
                return(ERROR);
}

void PrintLeafs(BTREE btree)
{
}
```

```
void PrintWinnerGames(BTREE btree)
{
}

int CountTotalSets(BTREE btree)
{
}

int CountWinnerSets(BTREE btree, void * winner)
{
}

void PrintAllGames(BTREE btree)
{
}
```