1. Implement the two functions without any c++11 or above features:

```
float GetSum(std::list< float > numbers)
{
   // todo
}

std::string GetName(std::map< int, std::string > names, int id)
{
   // todo
}
```

Please also write down if there are any changes you would propose?

2. Implement the functions of class **PlayerManager.**
   Don't change the **Player** struct and don't use any stl container for Player storage. You can add additional member variables to the PlayerManager.

```
struct Player
{
        char* name;
        int id;
};

class PlayerManager
{
public:
        PlayerManager() : numPlayers(0) {}
        ~PlayerManager();

        // destroys all players and deallocates their memory
        void DestroyAllPlayers();

        // allocates memory and returns the new player object
        Player* CreatePlayer(const char* name, int id);

        // destroys the player object and deallocates its memory
        void DestroyPlayerById(int id);
        void DestroyPlayer(Player* pPlayer);

        // returns the number of currently allocated players
        int GetNumPlayers();

        // returns a player by ID
        Player* GetPlayerById(int id);

private:
        Player** pPlayers;
        int numPlayers;

        // you can add additional member variables and functions if you need
};
```

Please also write down if there are any changes you would propose?

3. Implement the following two functions:

```
struct Vector
{
        float x, z, y;
};

float GetDistance(Vector a, Vector b)
{
   // todo
}

Vector GetClosestVector(Vector from, std::list< Vector > vectors)
{
   // todo
}
```

Please also write down if there are any changes you would propose?

4. Implement the three functions in BinarySearchTree

```
class BinarySearchTree
{
public:
        bool add(int value);
        bool remove(int value);
        bool contains(int value) const;

private:
        struct Node
        {
                int value;
                Node* left{ nullptr };
                Node* right{ nullptr };
        };

        Node* root{ nullptr };
};


bool BinarySearchTree::add(int value)
{
   // todo
}

bool BinarySearchTree::remove(int value)
{
   // todo
}

bool BinarySearchTree::contains(int value) const
{
   // todo
}
```

5. Implement the CMatchmaking class:

```
#include <functional>
#include <string>
#include <assert.h>

typedef int TRequestId;
#define INVALID_REQUEST 0

enum class EMatchState : int
{
        EWaiting,
        EFoundMatch,
        ENoMatchFound
};

// EWaiting: regularly reported while waiting
struct SMatchRequestState_Waiting
{
        int numberInQueue{ 0 };
};

// EFoundMatch: reported when match was found
struct SMatchRequestState_MatchFound
{
        std::string hostname;
        int port;
};

// ENoMatchFound: reported when no match was found
struct SMatchRequestState_MatchNotFound
{
        std::string reason;
};

// matchmaking service
struct IMatchmakingService
{
        // access to the matchmaker service instance
        static IMatchmakingService* GetService();

        // function to request a match
        // will call callback while request is active to notify about state
        typedef std::function<void(EMatchState state, void* state_data)> TCallback;
        virtual TRequestId RequestMatch(const TCallback& cb) = 0;

        // cancel pending match request.
        // No further callbacks are called and the request got cancelled
        virtual void CancelMatchRequest(TRequestId id) = 0;
};
```

```cpp
// matchmaking UI
struct IMatchmakingUI
{
        // update the match request dialog status message
        virtual void SetMatchSearchState(const char* message) = 0;

        // call one of the two functions to end the dialog
        virtual void OnMatchFound(const char* host, int port) = 0;
        virtual void OnMatchNoFound(const char* reason) = 0;

        // callback for user 'cancel' button on dialog
        // when called, the UI will close and no other action is needed
        typedef std::function<void()> TUserCancelCb;
        virtual void SetUserCb(const TUserCancelCb& cb) = 0;
        virtual void ClearUserCb() = 0;
};

// Please implement the following class
//  - request a match from the IMatchmakingService
//  - update the UI of the match request state while request is active
//  - call OnMatchFound/OnMatchNoFound once the matchmaking service has a result
//  - handle user callback to cancel the pending request
//  - cleanup properly
//  - validate input parameters and class usage
class CMatchmaking
{
public:
        CMatchmaking();
        ~CMatchmaking();

        // called by UI, user pressed 'search match' button
        void StartMatchRequest(IMatchmakingUI* pUI);

private:
        // you can add member variables and functions
};

CMatchmaking::CMatchmaking()
{
   // todo
}

CMatchmaking::~CMatchmaking()
{
   // todo
}

void CMatchmaking::StartMatchRequest(IMatchmakingUI* pUI)
{
   // todo
}
```

Please also write down if there are any changes you would propose?