Q2) To implement RPC using (Client)

#include <iostream>

#include "../Example1/Example1.h"

int main()

{

RPC\_STATUS status;

unsigned char\* szStringBinding = NULL;

// Creates a string binding handle.

// This function is nothing more than a printf.

// Connection is not done here.

status = RpcStringBindingCompose(

NULL, // UUID to bind to.

reinterpret\_cast<unsigned char\*>("ncacn\_ip\_tcp"), // Use TCP/IP

// protocol.

reinterpret\_cast<unsigned char\*>("localhost"), // TCP/IP network

// address to use.

reinterpret\_cast<unsigned char\*>("4747"), // TCP/IP port to use.

NULL, // Protocol dependent network options to use.

&szStringBinding); // String binding output.

if (status)

exit(status);

// Validates the format of the string binding handle and converts

// it to a binding handle.

// Connection is not done here either.

status = RpcBindingFromStringBinding(

szStringBinding, // The string binding to validate.

&hExample1Binding); // Put the result in the implicit binding

// handle defined in the IDL file.

if (status)

exit(status);

RpcTryExcept

{

// Calls the RPC function. The hExample1Binding binding handle

// is used implicitly.

// Connection is done here.

Output("Hello Implicit RPC World!");

}

RpcExcept(1)

{

std::cerr << "Runtime reported exception " << RpcExceptionCode()

<< std::endl;

}

RpcEndExcept

// Free the memory allocated by a string.

status = RpcStringFree(

&szStringBinding); // String to be freed.

if (status)

exit(status);

// Releases binding handle resources and disconnects from the server.

status = RpcBindingFree(

&hExample1Binding); // Frees the implicit binding handle defined in

// the IDL file.

if (status)

exit(status);

}

// Memory allocation function for RPC.

// The runtime uses these two functions for allocating/deallocating

// enough memory to pass the string to the server.

void\* \_\_RPC\_USER midl\_user\_allocate(size\_t size)

{

return malloc(size);

}

// Memory deallocation function for RPC.

void \_\_RPC\_USER midl\_user\_free(void\* p)

{

free(p);

}