

#include <iostream>

using namespace std;

float Division(float num, float den)

{

    try {

        return (num / den);

    }

    catch (...) {

        cout << "Zero Division Error\n";

    }

}

class memoryError : public exception {

    long long int size = 0;

public:

    memoryError();

    memoryError(long long int size) {

        this->size = size;

    }

    const char\* what() const throw()

    {

        return "Memory Overflow! size="+size;

    }

};

void Memory(long long int size) {

    try {

        int\* a = (int\*)malloc(sizeof(int) \* size);

        if (a == NULL) {

            memoryError E(size);

            throw E;

        }

    }

    catch (exception& e) {

        cout << e.what() << endl;

    }

}

void dataTypeMismatch() {

    try {

        int x = atoi("123a");

    }

    catch (...) {

        cout << "Data Type Mismatch\n";

    }

}

void outOfBoundIndex(int index) {

    try {

        int arr[10];

        if (index >= 10 || index < 0)throw 0;

        arr[index] = 0;

    }

    catch (...) {

        cout << "Index Of Bound Error\n";

    }

}

int main()

{

    //part 1 -----------------------------------

    Division(5,0);

    cout << "After Zero Division error\n";

    Memory(10000000000000);

    cout << "After Memory overflow error\n";

    dataTypeMismatch();

    cout << "After Format Mismatch\n";

    outOfBoundIndex(12);

    cout << "After Index Out Of Bound\n";

    cout << "Exception Handling Accomplished!\n";

    //-----------------------------------------

    //part 2 ----------------------------------

    //exception in outer try block

    cout << "exception in outer try block\n";

    try {

        cout << "Outer Try Block\n";

        throw 0;

        try {

            cout << "Inner Try Block\n";

        }

        catch (int x) {

            cout << "Inner Catch Block\n";

        }

    }

    catch (int x) {

        cout << "Outer try Block\n";

    }

    //exception in inner try block

    cout << "exception in inner try block\n";

    try {

        cout << "Outer Try Block\n";

        try {

            cout << "Inner Try Block\n";

            throw 0;

        }

        catch (int x) {

            cout << "Inner Catch Block\n";

        }

    }

    catch (int x) {

        cout << "Outer try Block\n";

    }

    //no exception

    cout << "no exception\n";

    try {

        cout << "Outer Try Block\n";

        try {

            cout << "Inner Try Block\n";

        }

        catch (int x) {

            cout << "Inner Catch Block\n";

        }

    }

    catch (int x) {

        cout << "Outer try Block\n";

    }

    //rethrowing catched exception

    cout << "rethrowing catched exception\n";

    try {

        cout << "Outer Try Block\n";

        try {

            cout << "Inner Try Block\n";

            throw 0;

        }

        catch (int x) {

            cout << "Inner Catch Block\n";

            throw x;

        }

    }

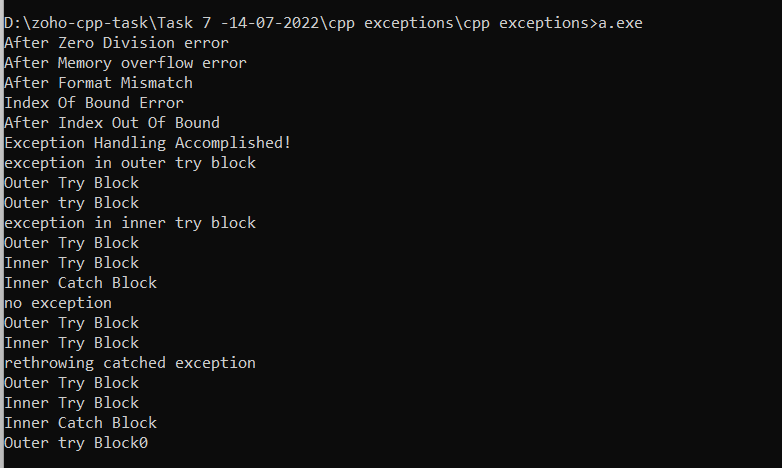
    catch (int x) {

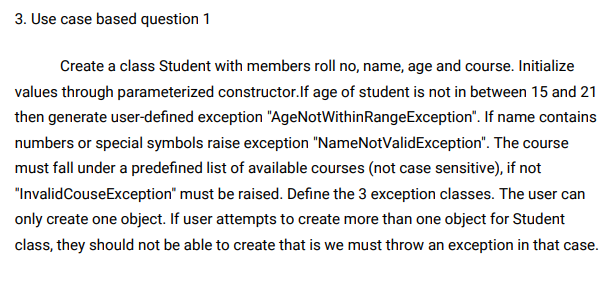
        cout << "Outer try Block"<<x<<endl;

    }

    //part 3 --------------------------------

}





#include <bits\stdc++.h>

using namespace std;

class AgeNotWithinRangeException : exception{

    public:

        const char\* what() const throw()

        {

            return "Age Not Within Range 15-21\n";

        }

};

class NameNotValidException : exception{

    public:

        const char\* what() const throw()

        {

            return "Name should Not contain numbers or special characters\n";

        }

};

class InvalidCourseException : exception{

    public:

        const char\* what() const throw()

        {

            return "Choose available courses only!\n";

        }

};

class MoreThanOneObject : exception{

    public:

        const char\* what() const throw()

        {

            return "Student already Created!\n";

        }

};

vector<string> courses{"python","cpp","java"};

class Student{

    int roll\_no;

    string name;

    int age;

    string course;

    public:

    static bool created;

    Student(int roll\_no,int age,string course,string name)

    :roll\_no(roll\_no),age(age),course(course),name(name)

    {

        if(!created)created = true;

        else{

            MoreThanOneObject e;

            throw e;

        }

        if(!(age<21 && age>15)){

            AgeNotWithinRangeException e;

            throw e;

        }

        for(int i=0;i<name.size();i++){

            if(isalpha(name[i]) || name[i]==' ')continue;

            NameNotValidException e;

            throw e;

        }

        transform(course.begin(), course.end(), course.begin(), ::tolower);

        if(find(courses.begin(),courses.end(),course)==courses.end()){

            InvalidCourseException e;

            throw e;

        }

        cout<<"New Student Created-"<<name<<endl;

    }

};

bool Student::created = false;

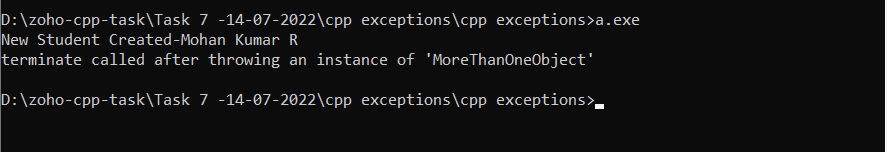
int main(){

    Student student1(59,20,"cpp","Mohan Kumar R");

    Student student2(50,20,"java","Logeshavan R");

    return 0;

}



**IOT System:**

Server.cpp

#include <iostream>

#include <ws2tcpip.h>

#include <unordered\_map>

#include <string>

#pragma comment (lib,"ws2\_32.lib")

using namespace std;

void printEnvironmentState(unordered\_map<string, string>& sensorStatus) {

    cout << " SENSORS   " << "STATUS \n";

    for (auto it : sensorStatus) {

        cout << it.first << "   " << it.second << endl;

    }

}

void main() {

    unordered\_map<string, string> sensorStatus;

    sensorStatus["fan"] = "off";

    sensorStatus["light"] = "off";

    sensorStatus["temperature"] = "0";

    sensorStatus["door"] = "close";

    while (true) {

        //Initialize winsock

        WSADATA wsData;

        WORD ver = MAKEWORD(2, 2);

        int wsOk = WSAStartup(ver, &wsData);

        if (wsOk != 0) {

            cerr << "Cant Initialize Winsock! Exiting" << endl;

            return;

        }

        else

        {

            cout << "Initialized Winsoc\n";

        }

        //Creating socket

        SOCKET listening = socket(AF\_INET, SOCK\_STREAM, 0);

        if (listening == INVALID\_SOCKET) {

            cerr << "Cant Create a Socket! Exiting" << endl;

            return;

        }

        else {

            cout << "....=====Created Socket=====....\n";

        }

        //Binding the socket to IP and port

        sockaddr\_in hint;

        hint.sin\_family = AF\_INET;

        hint.sin\_port = htons(54000);

        hint.sin\_addr.S\_un.S\_addr = INADDR\_ANY;

        bind(listening, (sockaddr\*)&hint, sizeof(hint));

        cout << "Bind IP and PORT to SOCKET\n";

        //telling winsock the socket is for listening

        listen(listening, SOMAXCONN);

        cout << "SOCKET in Listening mode\n";

        //wait for a connection

        sockaddr\_in client;

        int clientSize = sizeof(client);

        SOCKET clientSocket = accept(listening, (sockaddr\*)&client, &clientSize);

        if (clientSocket == INVALID\_SOCKET) {

            cerr << "Invalid Client Connection" << endl;

            return;

        }

        char host[NI\_MAXHOST];  //Clients remote name

        char service[NI\_MAXHOST];//port number the client is connected on

        ZeroMemory(host, NI\_MAXHOST);

        ZeroMemory(service, NI\_MAXHOST);

        if (getnameinfo((sockaddr\*)&client, sizeof(client), host, NI\_MAXHOST, service, NI\_MAXHOST, 0) == 0)

        {

            cout << host << "Connected on port " << service << endl;

        }

        else

        {

            inet\_ntop(AF\_INET, &client.sin\_addr, host, NI\_MAXHOST);

            cout << host << " connected on port " <<

                ntohs(client.sin\_port) << endl;

        }

        //close listening socket

        closesocket(listening);

        //while loop:accept and echo message back to client

        char buf[4096];

        ZeroMemory(buf, 4096);

        //Wait for client to send data

        int bytesReceived = recv(clientSocket, buf, 4096, 0);

        if (bytesReceived == SOCKET\_ERROR) {

            cerr << "Error in rev() Exiting!" << endl;

            break;

        }

        if (bytesReceived == 0)

        {

            cout << "Client Disconnected " << endl;

            break;

        }

        string sensor = string(buf, 0, bytesReceived);

        ZeroMemory(buf, 4096);

        bytesReceived = recv(clientSocket, buf, 4096, 0);

        string value = string(buf, 0, bytesReceived);

        if(sensor=="temperature") {

            sensorStatus[sensor] = value;

        }

        else {

            sensorStatus[sensor] = value;

        }

        //Closing the socket

        closesocket(clientSocket);

        //Cleaning up winsock

        WSACleanup();

        printEnvironmentState(sensorStatus);

        //Automation

        if (stof(sensorStatus["temperature"]) > 22) {

            cout << "IOT Central Automation - Turning on Fan due to high temperature\n";

            sensorStatus["fan"] = "on";

        }

        cout << endl << endl;

    }

}

Clients.cpp (sensors)

#include <iostream>

#include <string>

#include <WS2tcpip.h>

#include <chrono>

#include <thread>

#include <algorithm>

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

void sendMsg(SOCKET& sock, string userInput) {

    char buf[4096];

    if (userInput.size() > 0)       // Make sure the user has typed in something

    {

        // Send the text

        std::this\_thread::sleep\_for(std::chrono::milliseconds(50));

        int sendResult = send(sock, userInput.c\_str(), userInput.size() + 1, 0);

    }

}

void main(int argc, char\*\* argv)

{

    if (argc<2) {

        cout << "Specify On/Off Condtion as argument!\n";

        return;

    }

    string ipAddress = "127.0.0.1";         // IP Address of the server

    int port = 54000;                       // Listening port # on the server

    // Initialize WinSock

    WSAData data;

    WORD ver = MAKEWORD(2, 2);

    int wsResult = WSAStartup(ver, &data);

    if (wsResult != 0)

    {

        cout << "Can't start Winsock, Err #" << wsResult << endl;

        return;

    }

    // Create socket

    SOCKET sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock == INVALID\_SOCKET)

    {

        cout << "Can't create socket, Err #" << WSAGetLastError() << endl;

        WSACleanup();

        return;

    }

    // Fill in a hint structure

    sockaddr\_in hint;

    hint.sin\_family = AF\_INET;

    hint.sin\_port = htons(port);

    inet\_pton(AF\_INET, ipAddress.c\_str(), &hint.sin\_addr);

    // Connect to server

    int connResult = connect(sock, (sockaddr\*)&hint, sizeof(hint));

    if (connResult == SOCKET\_ERROR)

    {

        cout << "Can't connect to server, Err #" << WSAGetLastError() << endl;

        closesocket(sock);

        WSACleanup();

        return;

    }

    // Do-while loop to send and receive data

    //

    string sensor = argv[1];

    string val = argv[2];

    cout << "Sending sensor:" << sensor << " value:" << val << endl;

    transform(val.begin(), val.end(), val.begin(), ::tolower);

    sendMsg(sock, sensor);

    sendMsg(sock, val);

    //string userInput;

    // Gracefully close down everything

    closesocket(sock);

    WSACleanup();

}

