Q.1.1 CLR in C#:-

The Common Language Runtime (**CLR**), the virtual machine component of Microsoft's .NET framework, manages the execution of .NET programs. A process known as just-in-time compilation converts compiled code into machine instructions which the computer's CPU then executes.

2. Command Line compiler

Command line compiler takes C# code as input and transforms code to MSIL/CIL (Microsoft intermediate language/Common intermediate language).CSC.exe produces .dll (dynamic link library) and .exe (executable) files.

**Step 1:**  
  
Create a program first -> myprogram.cs (name is upto you :P) , so we can compile that using CSC.EXE .  
I have created a program containing simple code :

|  |  |
| --- | --- |
|  | using System;    public class myprogram  {    static void Main()   {    Console.WriteLine("I hate hello world");   }  } |

* Open the command prompt
* Reach at the palace where is your program file exists
* Locate the CSC (C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe )
* Give input to CSC (C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe  myprogram.cs)
* .exe will be generated ;) :)

3. JIT in Asp.Net framework:-

A Web Service or Web Forms file must be compiled to run within the CLR. Compilation can be implicit or explicit. Although you could explicitly call the appropriate compiler to compile your Web Service or Web Forms files, it is easier to allow the file to be complied implicitly. Implicit compilation occurs when you request the .asmx via HTTP-SOAP, HTTP-GET, or HTTP-POST. The parser (xsp.exe) determines whether a current version of the assembly resides in memory or in the disk. If it cannot use an existing version, the parser makes the appropriate call to the respective compiler (as you designated in the **Class** property of the .asmx page).

When the Web Service (or Web Forms page) is implicitly compiled, it is actually compiled twice. On the first pass, it is compiled into IL. On the second pass, the Web Service (now an assembly in IL) is compiled into machine language. **This process is called Just-In-Time JIT compilation** because it does not occurs until the assembly is on the target machine. The reason you do not compile it ahead of time is so that the specific JITter for your OS and processor type can be used. As a result, the assembly is compiled into the fastest possible machine language code, optimized and enhanced for your specific configuration. It also enables you to compile once and then run on any number of operating systems.

4. Sqlconnection class in Asp.Net :-

A **SqlConnection** object represents a unique session to a SQL Server data source. With a client/server database system, it is equivalent to a network connection to the server. **SqlConnection** is used together with SqlDataAdapter and **SqlCommand** to increase performance when connecting to a Microsoft SQL Server database.

5. Namespace in c#:-

They're used especially to provide the **C#** compiler a context for all the named information in your program, such as variable names. Without **namespaces**, for example, you wouldn't be able to make a class named Console, as .NET already uses one in its System **namespace**. ... Data in a **namespace** is referred to by using the Namespace.

Q.2.1 steps for Design website with Master Pages:-

Master page provides the layout and functionality to the other pages. Creating a master page in ASP.NET is very easy. Let's start creating master page step by step.

**Step 1: Open new project in visual studio**

New project->Installed->Web->ASP.NET Web Application

After clicking OK button in the Window, select Empty

After clicking OK button, project "masterpage" opens but no file is there

**Step 2:**Add new file in to our project.

Add the master page into our project.

Right click Project->Add->New item

After clicking on new item, Window will open, select Web Form->Web Forms Master Page

After clicking the add button, master page 'site1.master' adds to our project.

Click on site1.master into Solution Explorer

**Step 3:**Design the master page, using HTML.

**HTML code of my master page is,**

1. <%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs" Inherits="masterpage.Site1" %>
3. <!DOCTYPE html>
5. <html xmlns="http://www.w3.org/1999/xhtml">
6. <head runat="server">
7. <title>c# corner</title>
8. <link href="css/my.css" rel="stylesheet" />
9. <asp:ContentPlaceHolder ID="head" runat="server">
10. </asp:ContentPlaceHolder>
11. </head>
12. <body>
13. <!DOCTYPE html>
14. <html>
15. <head>
16. <title>my layout</title>
17. <link rel="stylesheet" type="text/css" href="my.css">
18. </head>
19. <body>
20. <header id="header">
21. <h1>c# corner</h1>
22. </header>
23. <nav id="nav">
24. <ul>
25. <li><a href="home.aspx">Home</a></li>
26. <li><a href="#">About</a></li>
27. <li><a href="#">Article</a></li>
28. <li><a href="#">Contact</a></li>
29. </ul>
30. </nav>
31. <aside id="side">
32. <h1>news</h1>
33. <a href="#"><p>creating html website</p></a>
34. <a href="#"><p>learn css</p></a>
35. <a href="#">learn c#</a>
36. </aside>

39. <div id="con">
40. <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
42. </asp:ContentPlaceHolder>
43. </div>

46. <footer id="footer">
47. copyright @c# corner
48. </footer>
49. </body>
50. </html>
51. <form id="form1" runat="server">
53. </form>
54. </body>
55. </html>

**CSS Code**

1. #header{
2. **color**: #247BA0;
3. **text-align**: center;
4. **font-size**: 20px;
5. }
6. #nav{
7. **background-color**:#FF1654;
8. **padding**: 5px;
9. }
10. ul{
12. **list-style-type**: none;
13. }
14. li a {
15. **color**: #F1FAEE;
16. **font-size**: 30px;
17. column-**width**: 5%;
18. }
19. li
20. {
21. **display**: inline;
22. **padding-left**: 2px;
23. column-**width**: 20px;
24. }
25. a{
26. **text-decoration**: none;
27. **margin-left**:20px
28. }
29. li a:hover{
30. **background-color**: #F3FFBD;
31. **color**: #FF1654;
32. **padding**:1%;
33. }
34. #side{
35. **text-align**: center;
36. **float**: right;
37. **width**: 15%;
38. **padding-bottom**: 79%;
39. **background-color**: #F1FAEE;
40. }
41. #article{
42. **background-color**: #EEF5DB;
43. **padding**: 10px;
44. **padding-bottom**: 75%;
45. }
46. #footer{
47. **background-color**: #C7EFCF;
48. **text-align**:center;
49. **padding-bottom**: 5%;
50. **font-size**: 20px;
51. }
52. #con{
53. **border**:double;
54. **border-color**:burlywood;
55. }

Our master page is designed. Move to the next step.

**Step 4: A**dd web form in to our project.

Right click on the project->Add->New item

Now, design our homepage.

Here, we write home page only,

**Home.aspx**

1. <%@ Page Title="" Language="C#" MasterPageFile="~/Site1.Master" AutoEventWireup="true" CodeBehind="home.aspx.cs" Inherits="masterpage.home" %>
2. <asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
3. </asp:Content>
4. <asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
5. <h1>Home page</h1>
6. </asp:Content>

Finally, our Master page is created; build and run the project.

The master page looks aas shown in the picture:

2. Validation Controls used in c#:-

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

* RequiredFieldValidator
* RangeValidator
* CompareValidator
* RegularExpressionValidator
* CustomValidator
* ValidationSummary

BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

|  |  |
| --- | --- |
| **Members** | **Description** |
| ControlToValidate | Indicates the input control to validate. |
| Display | Indicates how the error message is shown. |
| EnableClientScript | Indicates whether client side validation will take. |
| Enabled | Enables or disables the validator. |
| ErrorMessage | Indicates error string. |
| Text | Error text to be shown if validation fails. |
| IsValid | Indicates whether the value of the control is valid. |
| SetFocusOnError | It indicates whether in case of an invalid control, the focus should switch to the related input control. |
| ValidationGroup | The logical group of multiple validators, where this control belongs. |
| Validate() | This method revalidates the control and updates the IsValid property. |

RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

<asp:RequiredFieldValidator ID="rfvcandidate"

runat="server" ControlToValidate ="ddlcandidate"

ErrorMessage="Please choose a candidate"

InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>

RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Type | It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String. |
| MinimumValue | It specifies the minimum value of the range. |
| MaximumValue | It specifies the maximum value of the range. |

The syntax of the control is as given:

<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"

ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"

MinimumValue="6" Type="Integer">

</asp:RangeValidator>

CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Type | It specifies the data type. |
| ControlToCompare | It specifies the value of the input control to compare with. |
| ValueToCompare | It specifies the constant value to compare with. |
| Operator | It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck. |

The basic syntax of the control is as follows:

<asp:CompareValidator ID="CompareValidator1" runat="server"

ErrorMessage="CompareValidator">

</asp:CompareValidator>

RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

|  |  |
| --- | --- |
| **Character Escapes** | **Description** |
| \b | Matches a backspace. |
| \t | Matches a tab. |
| \r | Matches a carriage return. |
| \v | Matches a vertical tab. |
| \f | Matches a form feed. |
| \n | Matches a new line. |
| \ | Escape character. |

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

|  |  |
| --- | --- |
| **Metacharacters** | **Description** |
| . | Matches any character except \n. |
| [abcd] | Matches any character in the set. |
| [^abcd] | Excludes any character in the set. |
| [2-7a-mA-M] | Matches any character specified in the range. |
| \w | Matches any alphanumeric character and underscore. |
| \W | Matches any non-word character. |
| \s | Matches whitespace characters like, space, tab, new line etc. |
| \S | Matches any non-whitespace character. |
| \d | Matches any decimal character. |
| \D | Matches any non-decimal character. |

Quantifiers could be added to specify number of times a character could appear.

|  |  |
| --- | --- |
| **Quantifier** | **Description** |
| \* | Zero or more matches. |
| + | One or more matches. |
| ? | Zero or one matches. |
| {N} | N matches. |
| {N,} | N or more matches. |
| {N,M} | Between N and M matches. |

The syntax of the control is as given:

<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"

ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>

CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

<asp:CustomValidator ID="CustomValidator1" runat="server"

ClientValidationFunction=.cvf\_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>

ValidationSummary

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

* **ShowSummary** : shows the error messages in specified format.
* **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

<asp:ValidationSummary ID="ValidationSummary1" runat="server"

DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />

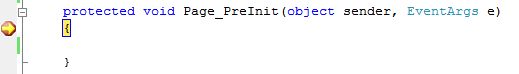
Validation Groups

Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

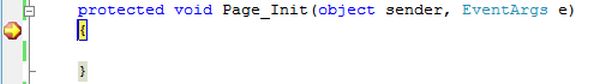
To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup*property.

Q.3 Page Life Cycle in Asp.Net

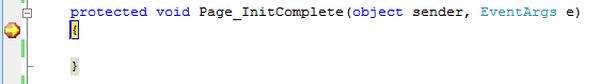
When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. The following are the various stages or events of ASP.Net page life cycle.  
  
**PreInit**

1. Check the IsPostBack property to determine whether this is the first time the page is being processed.
2. Create or re-create dynamic controls.
3. Set a master page dynamically.
4. Set the Theme property dynamically.  
     
   

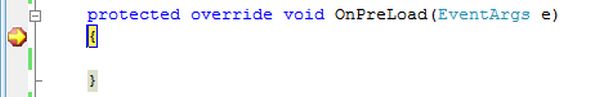
**Note**If the request is a postback then the values of the controls have not yet been restored from the view state. If you set a control property at this stage, its value might be overwritten in the next event.  
 **Init**

1. This event fires after each control has been initialized.
2. Each control's UniqueID is set and any skin settings have been applied.
3. Use this event to read or initialize control properties.
4. The "Init" event is fired first for the bottom-most control in the hierarchy, and then fired up the hierarchy until it is fired for the page itself.  
     
   

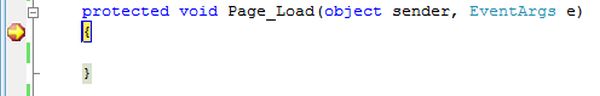
**InitComplete**

1. Until now the viewstate values are not yet loaded, hence you can use this event to make changes to the view state that you want to ensure are persisted after the next postback.
2. Raised by the Page object.
3. Use this event for processing tasks that require all initialization to be complete.  
     
   

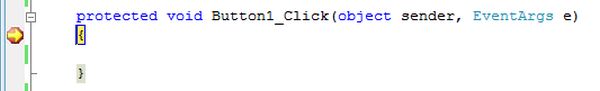
**OnPreLoad**

1. Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the Request instance.
2. Before the Page instance raises this event, it loads view state for itself and all controls, and then processes any postback data included with the Request instance.
3. Loads ViewState: ViewState data are loaded to controls.
4. Loads Postback data: Postback data are now handed to the page controls.  
     
   

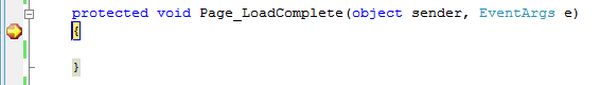
**Load**

1. The Page object calls the OnLoad method on the Page object, and then recursively does the same for each child control until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page.
2. This is the first place in the page lifecycle that all values are restored.
3. Most code checks the value of IsPostBack to avoid unnecessarily resetting state.
4. You may also call Validate and check the value of IsValid in this method.
5. You can also create dynamic controls in this method.
6. Use the OnLoad event method to set properties in controls and establish database connections.  
     
   

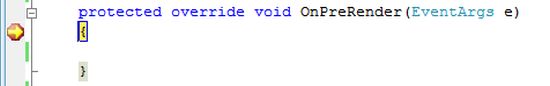
**Control PostBack Event(s)**

1. ASP.NET now calls any events on the page or its controls that caused the PostBack to occur.
2. Use these events to handle specific control events, such as a Button control's Click event or a TextBox control's TextChanged event.
3. In a postback request, if the page contains validator controls, check the IsValid property of the Page and of individual validation controls before performing any processing.
4. This is just an example of a control event. Here it is the button click event that caused the postback.  
     
   

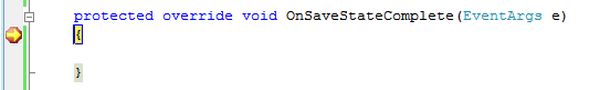
**LoadComplete**

1. Raised at the end of the event-handling stage.
2. Use this event for tasks that require that all other controls on the page be loaded.  
     
   

**OnPreRender**

1. Raised after the Page object has created all controls that are required in order to render the page, including child controls of composite controls.
2. The Page object raises the PreRender event on the Page object, and then recursively does the same for each child control. The PreRender event of individual controls occurs after the PreRender event of the page.
3. The PreRender event of individual controls occurs after the PreRender event of the page.
4. Allows final changes to the page or its control.
5. This event takes place before saving ViewState, so any changes made here are saved.
6. For example: After this event, you cannot change any property of a button or change any viewstate value.
7. Each data bound control whose DataSourceID property is set calls its DataBind method.
8. Use the event to make final changes to the contents of the page or its controls.  
     
   

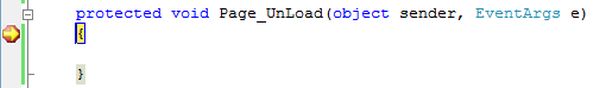
**OnSaveStateComplete**

1. Raised after view state and control state have been saved for the page and for all controls.
2. Before this event occurs, ViewState has been saved for the page and for all controls.
3. Any changes to the page or controls at this point will be ignored.
4. Use this event perform tasks that require the view state to be saved, but that do not make any changes to controls.  
     
   

**Render Method**

1. This is a method of the page object and its controls (and not an event).
2. The Render method generates the client-side HTML, Dynamic Hypertext Markup Language (DHTML), and script that are necessary to properly display a control at the browser.

**UnLoad**

1. This event is used for cleanup code.
2. At this point, all processing has occurred and it is safe to dispose of any remaining objects, including the Page object.
3. Cleanup can be performed on:  
   * Instances of classes, in other words objects
   * Closing opened files
   * Closing database connections.
4. This event occurs for each control and then for the page.
5. During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream.
6. If you attempt to call a method such as the Response.Write method then the page will throw an exception.  
     
   

**EXAMPLES**  
  
**Example 1: Control Values**  
  
In the following code, I have assigned the values to the label control on each event. When you run the code, you will see that in the "Page\_UnLoad", the values are not assigned to the label. WHY? Because, during the unload stage, the page and its controls have been rendered, so you cannot change the values.

Q.3.2 WAP to demonstrate the use of File Upload Wizard in c#

The ASP.Net FileUpload control is supported in .Net 2.0, 3.0, 3.5, 4.0 and 4.5 versions.

**HTML Markup**

The following HTML Markup consists of an ASP.Net FileUpload control, a Button and a Label control. The Button has been assigned a Click event handler.

1. <asp:FileUpload ID="FileUpload1" runat="server" />
2. <hr />
3. <asp:Button ID="btnUpload" Text="Upload" runat="server" OnClick="UploadFile" />
4. <br />
5. <asp:Label ID="lblMessage" ForeColor="Green" runat="server" />

8. **Namespaces**
9. You will need to import the following namespace.
10. **C#**
11. using System.IO;
13. **VB.Net**
14. Imports System.IO

**Create Folder (Directory) and Upload file in ASP.**

When the Upload Button is clicked, first a check is performed whether the Folder (Directory) exists. If it does not then the Folder (Directory) is created.

Then the uploaded File is saved into the Folder (Directory).

Finally a success message is displayed on the screen using the Label control.

**C#**

1. protected void UploadFile(object sender, EventArgs e)
2. {
3. string folderPath = Server.MapPath("~/Files/");
5. //Check whether Directory (Folder) exists.
6. if (!Directory.Exists(folderPath))
7. {
8. //If Directory (Folder) does not exists. Create it.
9. Directory.CreateDirectory(folderPath);
10. }
12. //Save the File to the Directory (Folder).
13. FileUpload1.SaveAs(folderPath + Path.GetFileName(FileUpload1.FileName));
15. //Display the success message.
16. lblMessage.Text = Path.GetFileName(FileUpload1.FileName) + " has been uploaded.";
17. }

Q.4.1 Session technique in Asp.Net

Session are the server side method of managing the state of an application i.e. all the web applications' state related info will be stored on server side if we use this technique. The benefit of having this technique is that since we are keeping all the state related information on server, the request and response becomes lightweight

The advantages of using Session State are

Better security

Reduced bandwidth

The disadvantages of using Session state are

More resource consumption of server.

Extra code/care if a Web farm is used(we will discuss this shortly)

Let us start looking into how we can use the Session state in our application and see how we can configure it to use various techniques.Before we proceed, let us see what all session management techniques are present in the ASP.NET framework.

In-Proc.

SQLServer.

StateServer.

How to configure Sessions

To configure the session management we need to specify the settings in the web.config file. Typical settings in web.config looks like:

<sessionState mode="InProc"

stateConnectionString="tcpip=127.0.0.1:42424"

sqlConnectionString="Data Source=.\SQLEXPRESS;Trusted\_Connection=Yes;"

cookieless="false"

timeout="100"/>

Using SQLServer Session Mode

If we use the SqlServer mode of session management then the session data will be stored in the SqlServer. The benefit of having this scenario is that the data is stored in a centralized database rather than the server memory. Let us see how this can be configured from web.config

<sessionState mode="SQLServer"

stateConnectionString="tcpip=127.0.0.1:42424"

sqlConnectionString="Data Source=.\SQLEXPRESS;Trusted\_Connection=Yes;"

cookieless="false"

timeout="100"/>

Apart from this the SQLServer also needs to be configured to store the session data. To do that simply type aspnet\_regsql on Visual studio command prompt and a configuration wizard will open. Here we can configure the SqlServer to cater the session data. The demo project has sqlexpress in local machine configured for this. To run the application, you will have to configure the sqlserver on local machine and perhaps change the web.config of the website.This way the server memory is not affected by the size of session data

Using this mode will only involve the overhead of additional database access to retrieve the session data. But there is one important aspect to remember when we are planning to use this technique. We need to send and receive the data to a SqlServer database, which will be a separate server. To successfully send and receive the data to SqlServer all the data that is being kept in the Session has to be serializable.

### Using StateServer Session Mode

If we use the StateServer mode of session management then the session data will be stored in a separate computer(server) and the session data will be handled by a windows service. The benefit of having this scenario is that the data is stored in a centralized location i.e. a state server rather than the individual server memory. Let us see how this can be configured from web.config

<sessionState mode="StateServer"

stateConnectionString="tcpip=127.0.0.1:42424"

sqlConnectionString="Data Source=.\SQLEXPRESS;Trusted\_Connection=Yes;"

cookieless="false"

timeout="100"/>

Apart from this the Server that will be used as state server also needs to be configured to store the session data. To do that simply run the service named ASP.NET state service to enable the desired Server to cater the session data. The demo project does the same on local machine configured for this. To run the application, you will have to run the service on local machine and perhaps change the web.config of the website.

Q.4.2 WAP to calculate the square, cube and fourth using Radio Button

using System;

public class Calculator

{

public static void Main()

{

int i,ctr;

Console.Write("\n\n");

Console.Write("Display the cube of the number:\n");

Console.Write("---------------------------------");

Console.Write("\n\n");

Console.Write("Input number of terms : ");

ctr= Convert.ToInt32(Console.ReadLine());

Console.Write("Squre of given number : ");

for(i=1;i<=ctr;i++)

{

Console.Write("Number is : {0} and cube of the {1} is :{2} \n",i,i, (i\*i));

}

for(i=1;i<=ctr;i++)

{

Console.Write("Cube of given number : ");

Console.Write("Number is : {0} and cube of the {1} is :{2} \n",i,i, (i\*i\*i));

}

Console.Write("Fourth of given number : ");

for(i=1;i<=ctr;i++)

{

Console.Write("Number is : {0} and cube of the {1} is :{2} \n",i,i, (i\*i\*i\*i));

}

}

}