Control Statements in C#

Control statements in C# are fundamental building blocks that dictate the flow of control in a program. They enable the execution of specific blocks of code based on conditions, loops, and branching. Here are the primary control statements in C#:

1. Conditional Statements

if

Executes a block of code if a specified condition is true.

```csharp
if (condition)
{
   // code to be executed if condition is true
}
```

if-else

Executes one block of code if a condition is true, and another block if it is false.

```csharp
if (condition)
{
   // code to be executed if condition is true
}
else
{
   // code to be executed if condition is false
}
```

else if

Checks multiple conditions in sequence.

```csharp
if (condition1)
{
   // code to be executed if condition1 is true
}
else if (condition2)
{
    // code to be executed if condition2 is true
}
else
{
    // code to be executed if none of the conditions are true
}
```

switch

Selects one of many code blocks to be executed.

```csharp
switch (variable)
{
    case value1:
        // code to be executed if variable equals value1
        break;
    case value2:
        // code to be executed if variable equals value2
        break;
    default:
        // code to be executed if variable doesn't match any case
        break;
}
```

2. Loop Statements

for

Repeats a block of code a specified number of times.

```csharp
for (initialization; condition; iteration)
{
    // code to be executed
}
```

foreach

Iterates over a collection or array.

```csharp
foreach (type variable in collection)
{
    // code to be executed for each element in the collection
}
```

while

Repeats a block of code as long as a specified condition is true.

```csharp
while (condition)
{
    // code to be executed
}
```

do-while

Executes a block of code once, and then repeats it as long as a specified condition is true.

```csharp
do
{
    // code to be executed
} while (condition);
```

3. Jump Statements

break

Exits the nearest enclosing loop or switch statement.

```csharp
break;
```

continue

Skips the current iteration of the nearest enclosing loop and proceeds with the next iteration.

```csharp
continue;
```

return

Exits a method and optionally returns a value.

```csharp
return value; // value is optional
```

goto

Transfers control to a labeled statement.

```csharp
goto label;
// ...
label:
// code to be executed after the goto statement
```

throw

Throws an exception.

```csharp
throw new Exception("Error message");
```

4. Exception Handling Statements

try-catch

Handles exceptions that occur during the execution of a block of code.

```csharp
try
{
   // code that may throw an exception
}
catch (ExceptionType e)
{
   // code to handle the exception
}
```

try-catch-finally

Ensures that a block of code is executed regardless of whether an exception is thrown.

```csharp
try
{
   // code that may throw an exception
}
catch (ExceptionType e)
{
   // code to handle the exception
}
finally
{
   // code to be executed regardless of an exception
}
```

try-finally

Ensures that a block of code is executed after the try block, regardless of whether an exception is thrown.

```csharp
try
{
   // code that may throw an exception
}
finally
{
   // code to be executed regardless of an exception
}
```