Control Structures

if

syntax:

```
if(boolean expression)
{
    body of the if statement
}
```

When the statement is executed, the boolean expression is evaluated. If it is true the body is executed else it is skipped over. If there are no { }, the if applies to the next single statement.

Examples:

```
if(X<5)
{
    Z = Z + X;
    Y = Y – X;
}
```

Compare this to:

```
if(X<5)
    Z = Z + X;
    Y = Y – X;
```

What, if any, is the difference?

if - else

```
if(boolean expression)
{
        body to do if the expression is true
}
else
{
        body to do if the expression is false
}
```

One and only one of these two blocks will be executed.

if – else – if – else etc.

Example

```
if(G >= 90)
   ch = ‘A’;
else
   if(G >= 80)
      ch = ‘B’;
   else
      ch = ‘C’;
```

In the previous example there are three ranges of numbers being considered. This could also have been written as:

```
if(G>=90)
    ch = ‘A’;
if(G>=80 && G < 90)
    ch = ‘B’;
if(G < 80)
    ch = ‘C’;
```

while

syntax:

```
while(boolean expression)
{
        body of loop to be iterated
}
```

This is a pretest loop. The boolean expression is evaluated. If true execute the loop. Check the expression after each iteration of the loop and continue to iterate as long as the expression is true.

What does the following do?

```
sum = 0;
i = 1;
while(i <= 10)
{
        sum = sum + i;
        i = i + 1;
}
```

Compare previous to:

```
while(i <= 10)
        sum = sum + i;
        i = i + 1;
```

What if any is the difference; i.e., what happens when the { } are missing? Don't be fooled by the fact that the two statements are indented under the while.

There is a post-test version of while.

syntax:

```
do
{
        body of loop to be iterated
}while(boolean expression);
```

The condition is tested at the end of the loop rather than at the beginning. What is the one thing you can say about a do-while that you may not always be able to say about the while?

Problem:

Write the example on slide 8 as an equivalent do – while structure.

Can this always be done?

Is it always a matter of just rearranging the structure and moving the conditional to the end? If not, give an example showing that other changes must be made.

Another Challenge:

Suppose you have written a block of code that is interactive with a user and executes one time, and you are fairly certain it is correct. Add structure to this block so that it will always be done once, but before exiting, it will ask the user if she would like to do it again. If the answer is yes, do it again; otherwise, go on with the remainder of the program.

We'll draw a picture on how to do this.

Then we will look at an example:

ExampleRepeat.cs

for

syntax:

```
for(initializer; bool expression; updater)
{
        body of loop
}
```

Example

```
int i;
int sum = 0;
for(i=1; i<=10; i++)
{
        sum = sum + i;
}
```

Do a “flow chart” for this.

for flow of control

The initializing statement is done. Once it is done, it is not visited again.Then it becomes similar to a while as shown by the following pseudo code.

```
while(bool expression is true){
        do the body of the loop
        do the update
}
```

Primary Use of a for

The for structure is normally used when you want to execute a loop a predetermined number of times, often indexed with some integer counter.

We will look at a number of examples.

Some Problems

1. Given a positive integer N, find all the numbers greater than 1 that divide N.
2. Given a positive integer N, find the sum of all the even integers less than N.
3. Given a string St which has been initialized, count the number of times the letter e or E occurs in the string.
4. Given a string St that has been initialized, find out whether or not the word “earth” occurs in the string. If it does report the location of the first letter

Question

Consider the following C# statement code:

```
int i = 1;
for(  ;  ;  )
{
        Console.Write(i + “  “);
        i = i+1;
}
```

Will it compile? What happens? Try it.

After the i = i+1; statement, add the following:

```
if(i == 100)
    break;
```

Recompile and run it. Can you guess what the break command does? In what other structures might you use it?