**do...while Repetition Statement**

The *do...while* repetition statement is similar to the *while* statement.

```c
do
{
    statement
} while ( condition );
```

In the *while* statement, the loop-continuation condition test occurs at the beginning of the loop before the body of the loop executes.

The *do...while* statement tests the loop-continuation condition after the loop body executes; therefore, the loop body always executes **at least once**.

When a *do...while* terminates, execution continues with the statement after the *while* clause. Note that it is not necessary to use braces in the *do...while* statement if there is only one statement in the body; however, most programmers include the braces to avoid confusion between the while and do...while statements.
For example,

```c
#include "stdio.h"
#include "conio.h"
int main()
{
    int counter = 1;
    do
    {
        printf("%d",counter);
        counter++;                     
    } while ( counter <= 10 );
    getch();
    return 0;
}
```
In addition to the selection and repetition statements, C++ provides statements `break` and `continue` to alter the flow of control. The preceding section showed how `break` can be used to terminate a `switch` statement's execution. This section discusses how to use `break` in a repetition statement.

break and continue Statements

In addition to the selection and repetition statements, C++ provides statements `break` and `continue` to alter the flow of control. The preceding section showed how `break` can be used to terminate a `switch` statement's execution. This section discusses how to use `break` in a repetition statement.

Example

Find the summation of numbers given by user until the user give up.

```c
#include"stdio.h"
#include"conio.h"
int main(){
    int sum=0;
    int number,c;
    do {
        printf("give an integer ");
        scanf("%d",&number);
        sum+=number;
        printf("continue(1), stop(0) ");
        scanf("%d",&c);
    } while ( c!=0 );
    printf("%d",sum);
    getch();
    return 0;
}
```
break Statement

The `break` statement, when executed in a `while`, `for`, `switch` statement, causes immediate exit from that statement.

Program execution continues with the next statement. Common uses of the `break` statement are to escape early from a `loop` or to skip the remainder of a `switch` statement.

```c
#include<stdio.h>
#include<conio.h>
int main(){
    int count;
    for (count = 1; count <= 10; count++) { 
        if ( count == 5 ) break; 
        // break loop only if x is 5
        printf("%d",count);
    }
    printf("\nBroke out of loop at count = %d", count);
    getch();
    return 0;
}
```
When the if statement detects that count is 5, the break statement executes.

This terminates the for statement, and the program proceeds to line 11 (immediately after the for statement), which displays a message indicating the value of the control variable that terminated the loop.

The for statement fully executes its body only four times instead of 10.

Note that the control variable count is defined outside the for statement header, so that we can use the control variable both in the body of the loop and after the loop completes its execution.

If we declare a variable in a block, we can not use it outside of block. For example

```
#include<stdio.h>
#include<conio.h>
int main(){
    //int count;
    for (int count = 1; count <= 10; count++) {
        if (count == 5) break;
        // break loop only if x is 5
        printf("%d",count);
    }
    printf("\nBroke out of loop at count = %d", count);
    getch();
    return 0;
}
```

There is an error in line 11 about variable count. Because we declare it in for block.
continue Statement

The `continue` statement, when executed in a `while`, `for` statement, skips the remaining statements in the body of that statement and proceeds with the next iteration of the loop.

In `while` statement, the loop-continuation test evaluates immediately after the continue statement executes. In the `for` statement, the increment expression executes, then the loop-continuation test evaluates.

```
#include "stdio.h"
#include "conio.h"
int main(){
  int count;
  for (count = 1; count <= 10; count++) {
    if (count == 5) continue; // if count is 5, skip remaining code in loop
    printf("%d",count);
  }
  printf("\nUsed continue to skip printing 5");
  getch();
  return 0;
}
```
Previous example uses the `continue` statement (line 7) in a `for` statement to skip the output statement (line 9) when `if` (line 7) determines that the value of `count` is 5.

When the continue statement executes, program control continues with the increment of the control variable in the `for` header (line 9) and loops five more times.

We stated that the `while` statement could be used in most cases to represent the `for` statement. The one exception occurs when the increment expression in the `while` statement follows the continue statement. In this case, the increment does not execute before the program tests the loop-continuation condition, and the `while` does not execute in the same manner as the `for`.

Example

Write a program to sum all positive number given by user until the user enter 0. (negative numbers will be ignored)

```c
#include "stdio.h"
#include "conio.h"
int main(){
    int sum=0;
    int number;
    do {
        printf("give an integer ");
        scanf("%d", &number);
        if (number<0) continue;
        if (number==0) break;
        sum+=number;
    } while (1);
    printf("%d", sum);
    getch();
    return 0;
}
```
or

```c
#include "stdio.h"
#include "conio.h"

int main()
{
    int sum=0;
    int number;
    do {
        printf("give an integer ");
        scanf("%d", &number);
        if (number<0) continue;
        sum+=number;
    } while (number!=0);
    printf("%d", sum);
    getch();
    return 0;
}
```

or

```c
#include "stdio.h"
#include "conio.h"

int main()
{
    int sum=0;
    int number;
    for (; ; ) {
        printf("give an integer ");
        scanf("%d", &number);
        if (number<0) continue;
        if (number==0) break;
        sum+=number;
    }
    printf("%d", sum);
    getch();
    return 0;
}
```