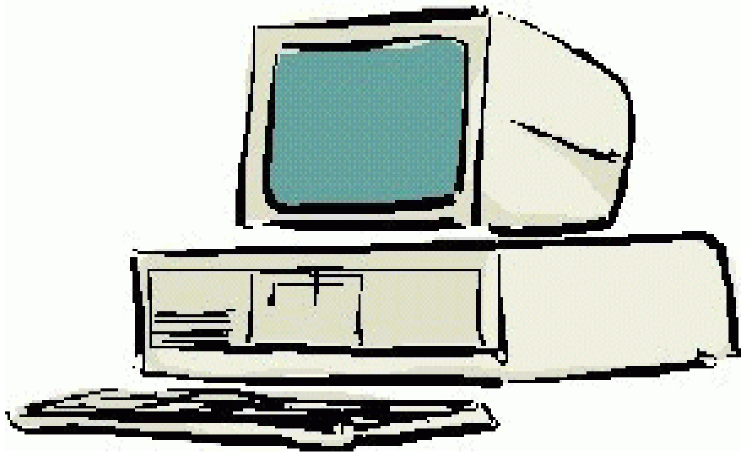
Programming in C



Session 4

Seema Sirpal
Delhi University Computer Centre

Looping

Loops can be created to execute a block of code for a fixed number of times.

Alternatively, loops can be created to repetitively execute a block of code until a boolean condition changes state. For instance, the loop may continue until a condition changes from false to true, or from true to false.

In this case, the block of code being executed must update the condition being tested in order for the loop to terminate at some point.

If the test condition is not modified somehow within the loop, the loop will never terminate. This creates a programming bug known as an infinite loop.

While

The while loop is used to execute a block of code as long as some condition is true.

If the condition is false from the start the block of code is not executed at all.

Its syntax is as follows.

```
while (tested condition is satisfied) {
    block of code
}
```

While

Here is a simple example of the use of while. This program counts from 1 to 100.

```
#include <stdio.h>
int main()
{
    int count = 1;

    while (count <= 100)
    {
        printf("%d\n“,count);
        count += 1; /* Shorthand for count = count + 1 */
    }
    return 0;
}
```

Do

The do loop also executes a block of code as long as a condition is satisfied.

The difference between a "do" loop and a "while" loop is that the while loop tests its condition at the top of its loop; the "do" loop tests its condition at the bottom of its loop.

If the test condition is false as the while loop is entered the block of code is skipped. Since the condition is tested at the bottom of a do loop, its block of code is always executed at least once. The "do" loops syntax is as follows

```
do {
   block of code
} while (condition is satisfied)
```

Do

<u>Example of the use of a do loop</u>:
The following program is a game that allows a user to guess a number between 1 and 100. A "do" loop is appropriate since we know that winning the game always requires at least one guess.

```
#include <stdio.h>
int main()
{
    int number = 44;
    int guess;
    printf("Guess a number between 1 and 100\n");
    do {
        printf("Enter your guess: ");
        scanf("%d",&guess);
        if (guess > number) {
            printf("Too high\n");
        }
        if (guess < number) {
            printf("Too low\n");
        }
    } while (guess != number);
    printf("You win. The answer is %d",number);
    return 0;
}
```

For

The third looping construct in C is the for loop.

The for loop can execute a block of code for a fixed number of repetitions. Its syntax is as follows.

```
for (initializations;test conditions;actions)
{
    block of code
}
```

For

The simplest way to understand for loops is to study several examples.

First, here is a for loop that counts from 1 to 10.

```
for (count = 1; count <= 10; count++)
{
    printf("%d\n",count);
}
```

For

The test conditions may be unrelated to the variables being initialized and updated (assigned).

Here is a loop that counts until a user response terminates the loop.

```
for (count = 1; response != 'N'; count++)
{
   printf("%d\n",count);
   printf("Continue (Y/N): \n");
   scanf("%c",&response);
}
```

For

More complicated test conditions are also allowed.

Suppose the user of the last example never enters "N", but the loop should terminate when 100 is reached, regardless.

```
for (count = 1; (response != 'N') && (count <= 100); count++)
{
   printf("%d\n",count);
   printf("Continue (Y/N): \n");
   scanf("%c",&response);
}
```

For

It is also possible to have multiple initializations and multiple actions.

This loop starts one counter at 0 and another at 100, and finds a midpoint between them.

```
for (i = 0, j = 100; j != i; i++, j--)
{
    printf("i = %d, j = %d\n",i,j);
}
printf("i = %d, j = %d\n",i,j);
```

For

Initializations are optional.

For instance, suppose we need to count from a user specified number to 100. The first semicolon is still required as a place keeper.

```
printf("Enter a number to start the count: ");
scanf("%d",&count);
for ( ; count < 100 ; count++)
{
    printf("%d\n",count);
}
```

For

The actions are also optional.

Here is a silly example that will repeatedly echo a single number until a user terminates the loop;

```
for (number = 5; response != 'Y';) {
    printf("%d\n",number);
    printf("Had Enough (Y/N) \n");
    scanf("%c",&response);
}
```

Practice

For practice, try implementing programs that will count down from 10 to 1 using while, do and for loops.

Solution using while

Solution using while

```
#include <stdio.h>
int main()
{

    int i = 10;

    while (i > 0)
    {
        printf("%d\n",i);
        i = i - 1;
    }

    return 0;
}
```

Solution using do

Solution using do

```
#include <stdio.h>

int main()
{

    int i = 10;

    do {
        printf("%d\n",i);
        i = i - 1;
    } while (i > 0);

    return 0;
}
```

Solution using for

Solution using for

```
#include <stdio.h>

int main()
{

    int i;

    for (i = 10; i > 0; i--)
    {
        printf("%d\n",i);
    }

    return 0;
}
```