CS1100 - Lecture 10

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

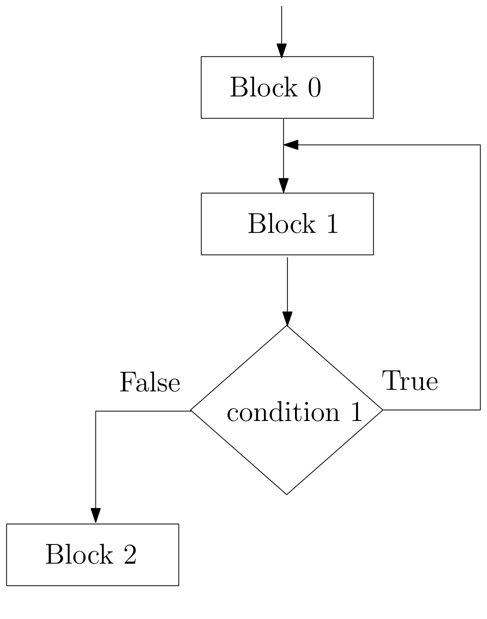
Usually programming languages have more than one type of loops available. This is done by the designers of the langauge to make the job of the programmers easier. Depending on the problem to solve, some type of loop may be more suitable than others. In previous classes, we studied about `while` loop and nested `while` loops. C supports two more structured loop constructs.

do-while loop

This is another type of loops permitted in C. The way of writing a `do-while` loop and its corresponding control flow diagram is shown in the figure below. Note that, control statements like `break` and `continue` can appear in `do-while` loops as well and the way they work is similar to that in the case of a `while` loop. However, we assume that, there are no `break` or `continue` statements in any of the program blocks in the example shown below.

```
Block 0
do
{
      Block 1
}while(condition 1);

Block 2
```



In this type of loop, the loop-condition is checked at the end of the loop. The difference between a `while` loop and a `do-while` loop is that, first one has an *entry* condition and the second one has an *exit* condition. Note that, according to the control flow diagram, Block 1 is executed at least once, even before checking `condition 1`. After Block 1 is executed, `condition 1` is evaluated and if `condition 1` evaluates to *true*, then program control will go back to the beginning of Block 1. The execution of Block 1 and evaluation of `condition 1` is repeated in this way, until `condition 1` becomes *false*.

Consider an example of a simple calculator, which repeatedly asks the user to enter a pair of numbers and computes the sum of the given pair and displays the result to the user. Each time after giving the result, the user should get an option to either stop the calculator or continue by taking another pair of numbers to find their sum. In such situations, a `do-while` loop is convenient.

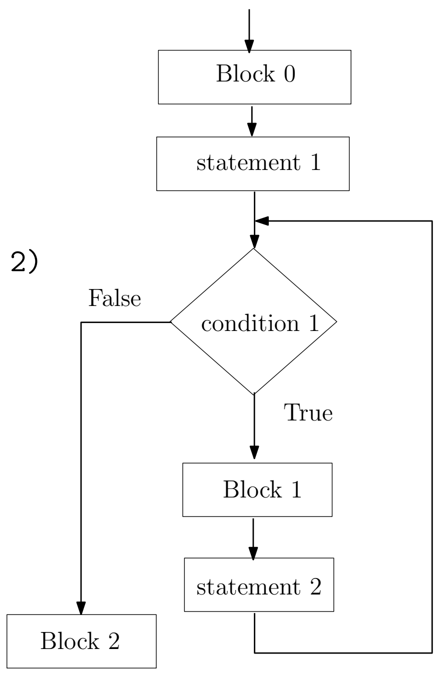
```
#include<stdio.h>
int main()
{
    int a, b, sum, option;
    printf("\nWelcome! Here is your simiple calculator\n\n");
    do
    {
       printf("Give two numbers to add \n");
       scanf("%d%d",&a,&b);
       sum=a+b;
       printf("sum is %d\n",sum);
       printf("Options 1: Do again, 0: Stop\n");
       scanf("%d",&option);
    } while(option==1);

    printf("\nGood Bye! Have a nice day!\n");
    return 0;
}
```

for loop

This is probably the most commonly used loop in C. The way of writing a `for` loop and its associated control flow is shown in the following figure. We assume that there are no `break` or `continue` statements in any of the program blocks in this figure, though in general `break` and `continue` can occur within `for` loops as well.

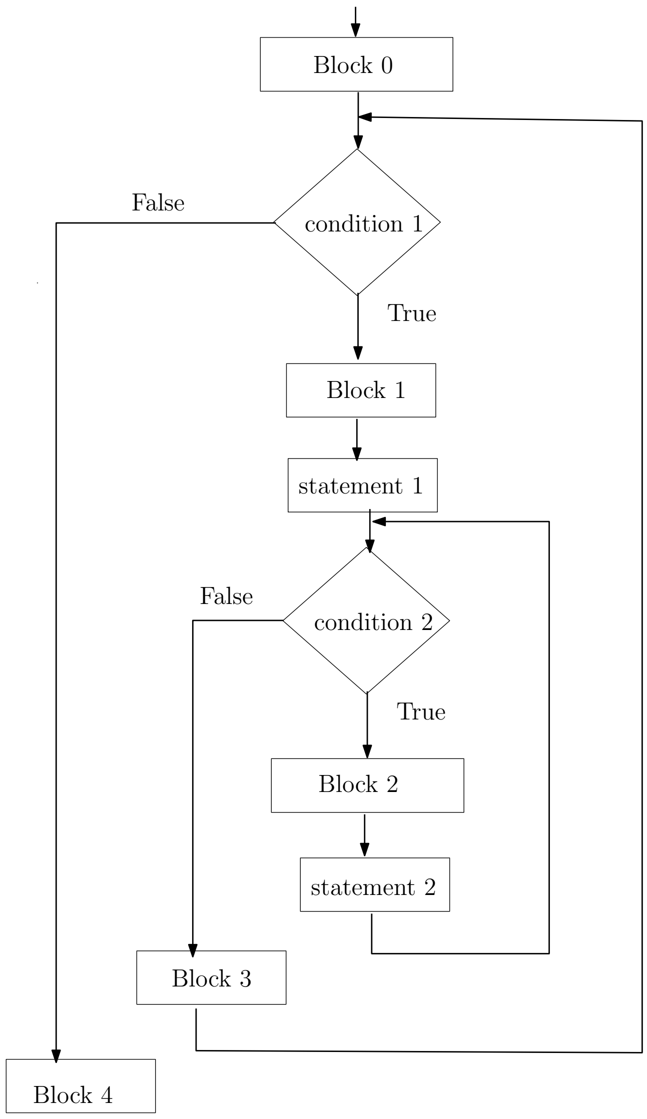
```
Block 0
for(statement 1; condition 1; statement 2)
{
    Block 1
}
Block 2
```



When control reaches the statement `for`, it will execute `statement 1` and after that the `condition 1` will be evaluated. If the `condition 1` evaluates to *true*, the program control will enter the loop and execute Block 1. After the execution of Block 1, it will execute `statement 2` and the control will go back to `condition 1`. This checking of the loop-condition, execution of Block 1 and `statement 2` is repeated until `condition 1` becomes *false* when it gets evaluated. Note that, `statement 1` (the statement before the first semi colon) is **not** executed repeatedly during this process. However, if this `for` loop is an inner loop within some other outer loop, then `statement 1` will be executed whenever the control flow proceeds from Block 0 to the `for` loop.

The control flow diagram of a for loop nested within a while loop is shown below. In this example, we assume that there are no break or continue statements in any of the program blocks.

```
Block 0
while(condition 1)
{
   Block 1
   for(statement 1;condition 2;statement 2)
   {
      Block 2
   }
   Block 3
}
Block 4
```

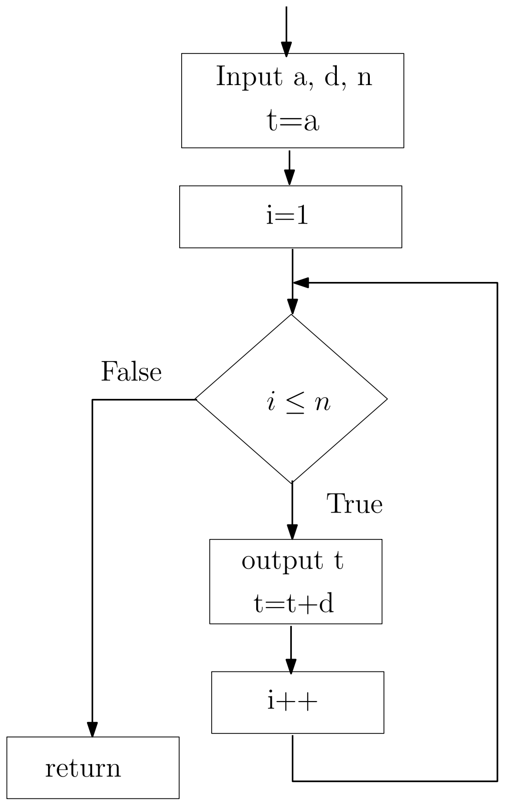


Examples of using for loop

The following program is for taking the first term and the common difference of an arithmetic progression and a number `n` and output the first `n` terms of the progression.

```
#include<stdio.h>
int main()
{
    int n, i, t, a, d;
    printf("Give first term and common difference of the AP \n");
    scanf("%d%d", &a,&d);
    printf("Give the number of terms you want \n");
    scanf("%d",&n);
    printf("first %d terms of the AP are\n", n);
    t=a;
    for (i=1; i<=n; i++)
    {
      printf("%d \n",t);
      t=t+d;
    }
   return 0;
}
```

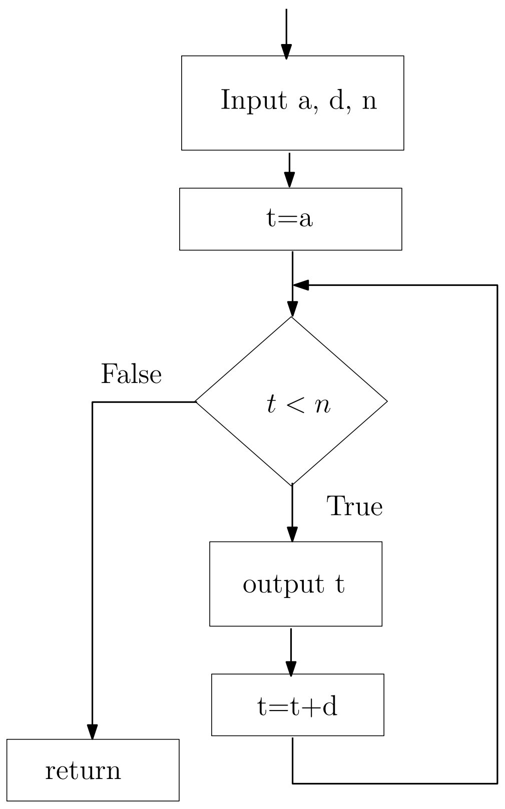
The control flow of the above program is shown below.



A program for taking the first term and the common difference of an arithmetic progression and a number `n` and output all terms which are less than `n` is given below.

```
#include<stdio.h>
int main()
{
    int n, i, t, a, d;
    printf("Give first term and common difference of the AP \n");
    scanf("%d%d", &a,&d);
    printf("Give the value of n \n");
    scanf("%d",&n);
    printf("terms less than %d are\n", n);
    for (t=a; t<n; t=t+d)
    {
      printf("%d \n",t);
    }
   return 0;
}
```

The control flow of the above program is shown below.



Structured and unstructured loops

The loops provided in programming languages like C are intended to make the control flow structured. If *loop 1* and *loop 2* are not nested loops, moving the program control from the middle of *loop 1* to the middle of *loop 2* should be avoided. Otherwise, the control flow becomes unstructured and difficult to follow.

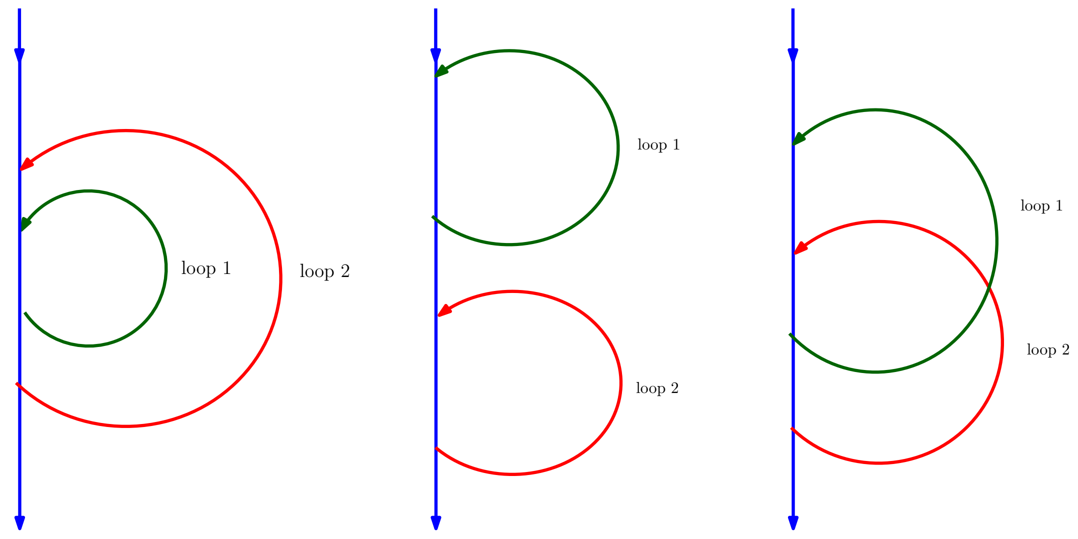


Figure 1: The first two figures show structured loops, whereas the third one is unstructured.

All the three types of loops we studied so far are examples of structured loop constructs. Structured programming langauges like C doesnt provide the programmer with a ready made easy to use unstructured loop construct. But still, there are some crude methods available in C which will permit a programmer to write unstructured code. Since such methods are advised to be avoided in modern programming, we will not study them in this course.