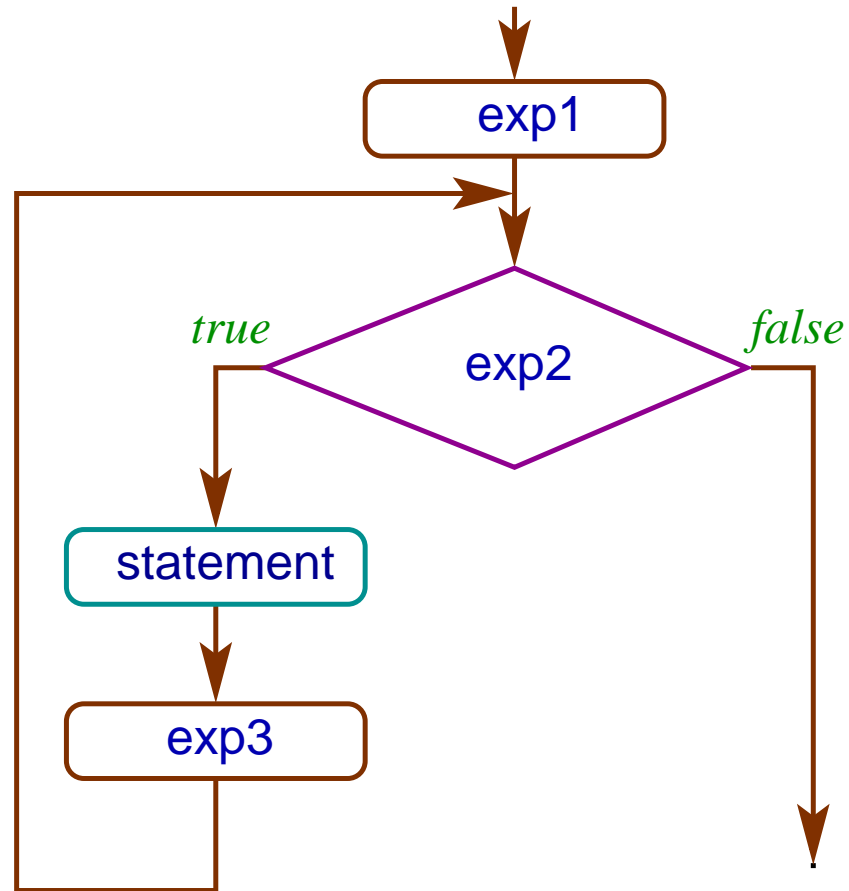


for Statement

Another *iterative* construct in C language is the **for**-statement (loop). The structure or the **syntax** of this statement is,

```
for (exp1; exp2; exp3) statement
```

for Loop



for Statement

- The **expression₁** is used for initialization before entering the loop.
- The **expression₂** is to control the loop, whether to enter the loop or to skip it.
- The **expression₃** executed after the execution of the **statement** part of the loop, is used essentially to update the loop control condition.

for Statement

- All three expressions can be omitted. The omitted control expression, **expression₂**, makes the condition **true**.
- The **statement** can also be null **‘;’**.

Example III

We write program to solve the problem of example III, the sum of first n natural numbers.

```
#include <stdio.h>
int main() // temp29.c
{
    int n, i, sum = 0;
    printf("Enter a +ve integer: ");
    scanf("%d", &n);
    for(i=1, sum=0; i<=n; ++i)
        sum += i;
    printf("0+ ... + %d = %d\n", n, sum);
    return 0;
}
```

for Statement

```
for (i=1, sum=0; i<=n; ++i)  
    sum += i;
```

expression₁

expression₁ is 'i=1, sum=0'. We have used the comma operator (,). This is evaluated left to right and is of lowest precedence among the operators. The value of the expression is the value of the rightmost operand. It is **sum = 0** in this case.

while and for Loops

A `while`-statement can be simulated by a `for`-statement.

$$\text{while}(\textit{exp}) \textit{stmt} \equiv \text{for}(\text{; } \textit{exp};) \textit{stmt}$$

while and for Loops

Similarly a `for`-statement can be simulated by a `while`-statement and expression statements.

```
for(exp1; exp2; exp3) stmt
```

≡

```
exp1; while(exp2) {stmt exp3;} 
```

This equivalence is not true if there is a `continue` statement in `stmt`.

Example IV

Read n int data and print the largest among them.

The first input is the number of data, n .
Following inputs are a sequence of n int data.
It is essential to print the input.

Inductive Definition

$$\text{lrgst}(d_1, d_2, \dots, d_n) = \begin{cases} d_1 & \text{if } n = 1, \\ \max(d_1, \text{lrgst}(d_2, \dots, d_n)) & \text{if } n > 1. \end{cases}$$

Sequence of Operations

- Read the number of data $n(\geq 1)$.
- Read the first data in a variable named **largest**^a.
- In a **for**-loop read the i^{th} data ($i = 2 \cdots n$).
If it is larger than the data present in **largest**, copy it to **largest**.

^aUpto this point of execution, this is the largest data.

```
#include <stdio.h>
int main() // temp30.c
{
    int n, largest, i;

    printf("Enter a +ve integer: ");
    scanf("%d", &n);
    printf("Enter %d integers:\n", n);
    scanf("%d", &largest);
    printf("Input data: %d ", largest) ;
    for(i=2; i<=n; ++i) {
        int temp ; // local to block

        scanf("%d", &temp);
```

```
        printf("%d ", temp) ;  
        if (temp > largest) largest = temp ;  
    }  
    printf("\nLargest: %d\n", largest);  
    return 0;  
}
```

Note

- It is not necessary to know the number of data **a priori**.
- We shall use **EOF** (end-of-file) from the keyboard (**Ctrl+D**) to terminate the input.
- Every call to **scanf()** returns the number of data read. If the **Ctrl+D** is pressed, **scanf()** returns a special constant **EOF** (end-of-file, defined in **stdio.h**).


```
#include <stdio.h>
int main() // temp31.c
{
    int largest, count = 0, temp;

    printf("Enter integer data\n");
    printf("Terminate by Ctrl+D\n");
    scanf("%d", &largest); // at least one data
    printf("Input data: %d ", largest) ;
    ++count ;
    for(; scanf("%d", &temp) != EOF;) {
        printf("%d ", temp);
        if (temp > largest) largest = temp ;
        ++count;
    }
}
```

```
}  
printf("\nLargest among %d data: %d\n",  
       count, largest);  
return 0;  
}
```