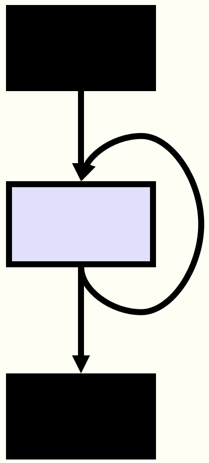
Looping

Repetition

Sometimes we want to repeat a block of code. This is called a *loop*.



- A “loop” is a repeated (“iterated”) sequence of statements
- Like conditionals, loops (iteration) give us a huge increase in the power of our programs
- **Alert**: loops are harder to master than `if` statements
  - Even experienced programmers often make subtle errors when writing loops

Motivating Loops

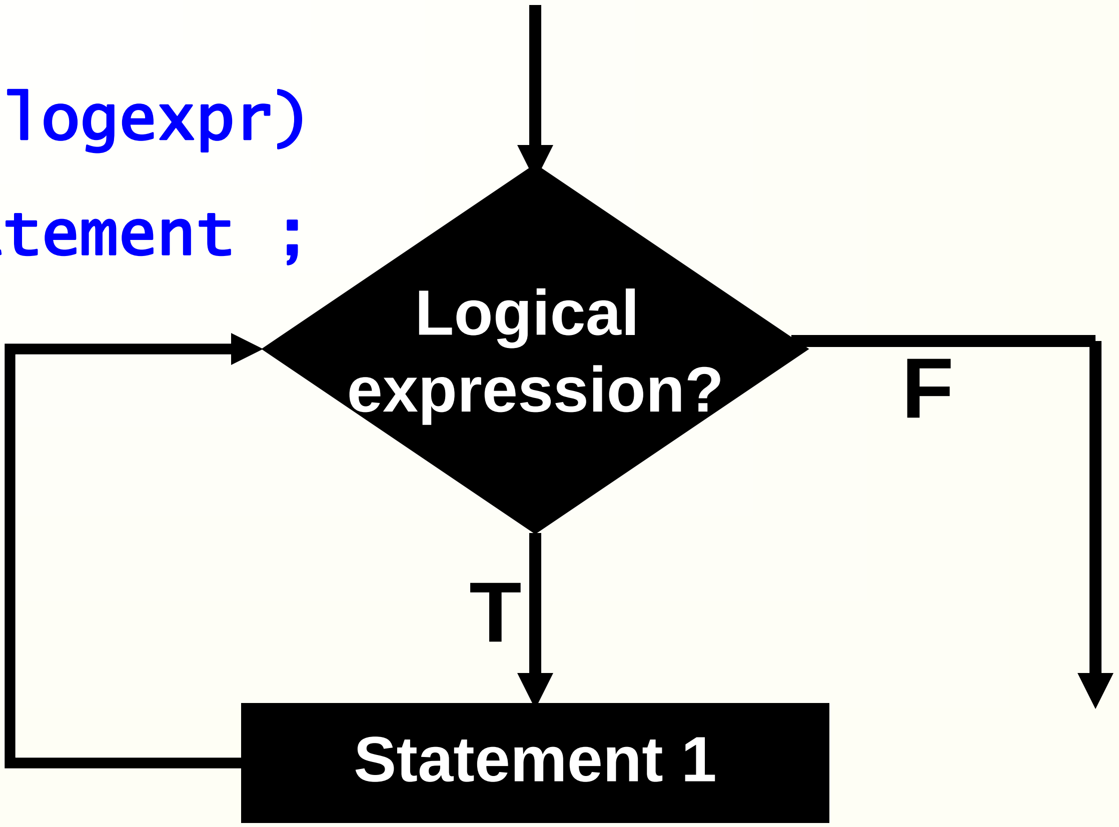
- **Problem: add 4 numbers entered at the keyboard.**

```
int  sum;
int x1, x2, x3, x4;
printf("Enter 4 numbers: ");
scanf("%d%d%d%d", &x1, &x2, &x3, &x4);
sum = x1 + x2 + x3 + x4;
```

- **This works perfectly!**
- **But... what if we had 14 numbers? or 40? or 4000?**

while Loop

```
while(logexpr)
     Statement ;
```



Example

```
void main()
{
     int k;
    k = 0;
    while(k<26)
           {
           printf("%c",k+'A');
           k++;
           }
}
```

while versus if

- Repeat a section of code depending on a condition
- Use a loop if you want to execute a section of code more than once

```
while (condition) {
      body of while
      }
```

Loops continuously until test is false

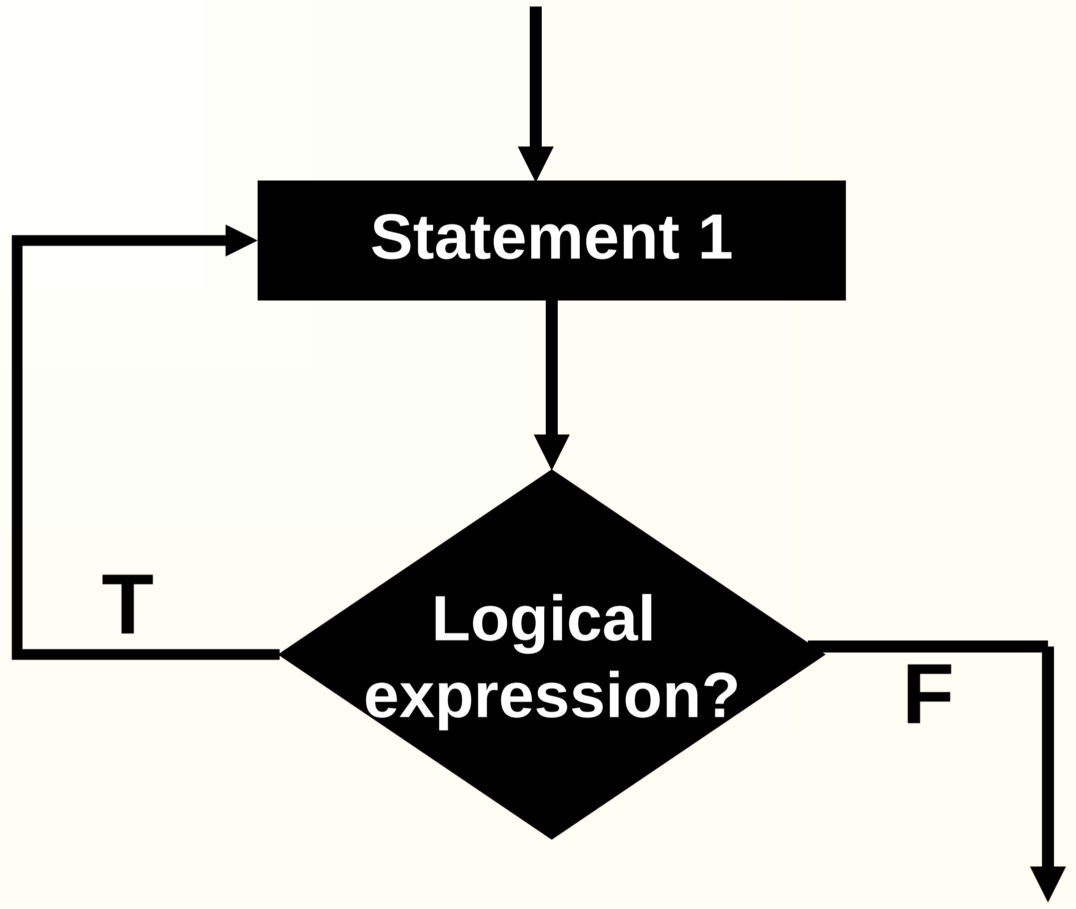
- May or may not execute a section of code, even if it does, it will be executed once
- Use the if statement when you wish to conditionally execute a section of code once

```
if (condition) {
      body of if
}
```

Executes once if test is true

do-while Loop

```
do
   statement
while(logexpr);
```



ALPHABET.C

Program to print uppercase alphabet.

```
#include <stdio.h>
int main()
{
  int k;
  k = 0;
  do
    {
     printf("%c",k+'A');
     k++;
    }
  while (k < 26);
  return 0;
}
```

OUTPUT

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

ISVALID.C

Program to force user to input a valid value between 1 and 100.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int num, isvalid;
    /* Accept and validate user input for num */
    do
        {
        printf("\nEnter an integer between 1 and 100 : ");
        scanf("%d",&num);
        isvalid = ( num >=1 && num <=100);
        if( !isvalid )
            printf("\nERROR: Invalid input, try again\n");
        }
    while(!isvalid);
    printf("\n\n Your valid integer input is %d",num);
    return 0;
}
```

do..while statement

- The body of the `do...while` loop executes **at least once**

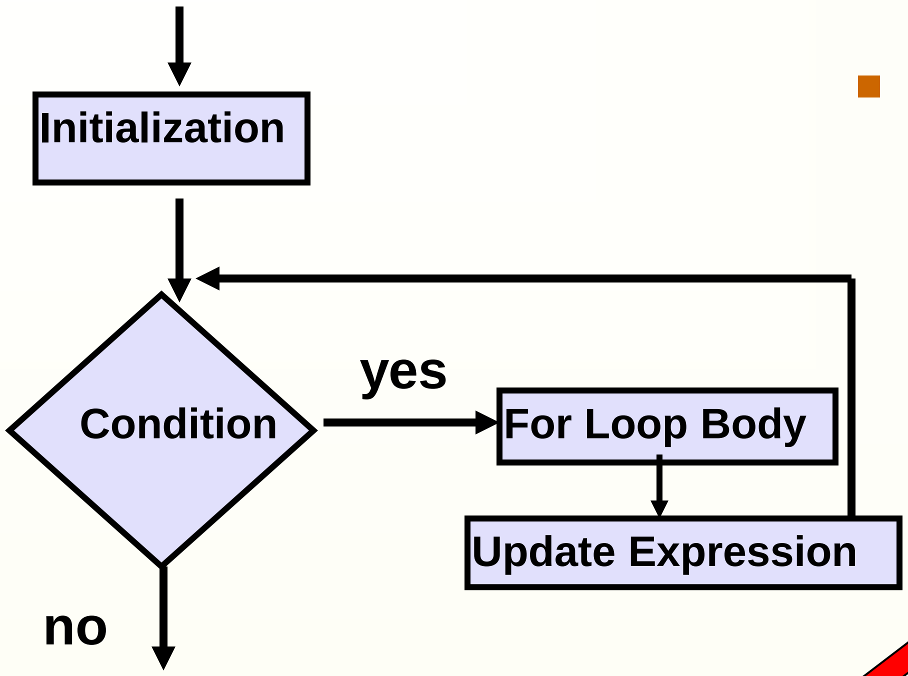
```
while ( count <= number_inputs )
{
        scanf("%d", &x);
        sum = sum + x;
        count = count + 1;
}
```

Relational test at the top

```
do {
        scanf("%d", &x);
        sum = sum + x;
        count = count + 1;
} while ( count <= number_inputs );
```

Relational test at the bottom

for loops



- Any for loop can be written as a while loop

```
initialization;
while (condition)
{
   statement;
   update;
}
```

```
for (initialization; condition; update)
   statement;
```

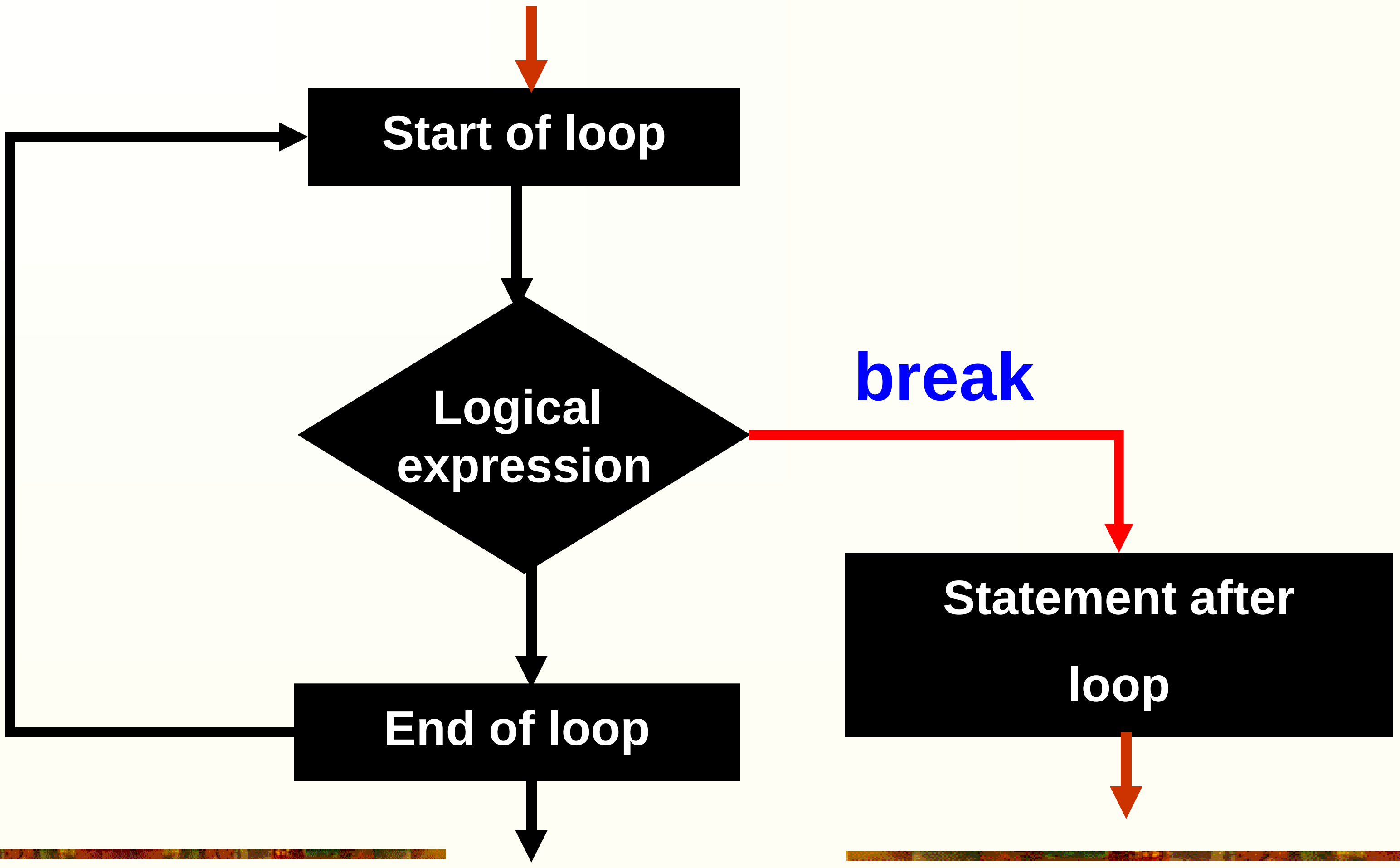
Point To Note

- Counting is not done as 1, 2, 3, ... but as 0, 1, 2, ... To become a C programmer, you must become familiar with this unusual counting scheme starting from **0** and going upto **(n-1)**.
- All three expressions in the for statement are optional.

  for(;logexpr;)

  - is exactly equivalent to a while statement.

break Statement



`break` Statement

- Instead of exiting an entire program, use the `break` statement to exit the current loop or section of code using the format:

```
break;
```

- You can use it anywhere, but it typically appears in the body of a loop or in a `switch` statement

Example

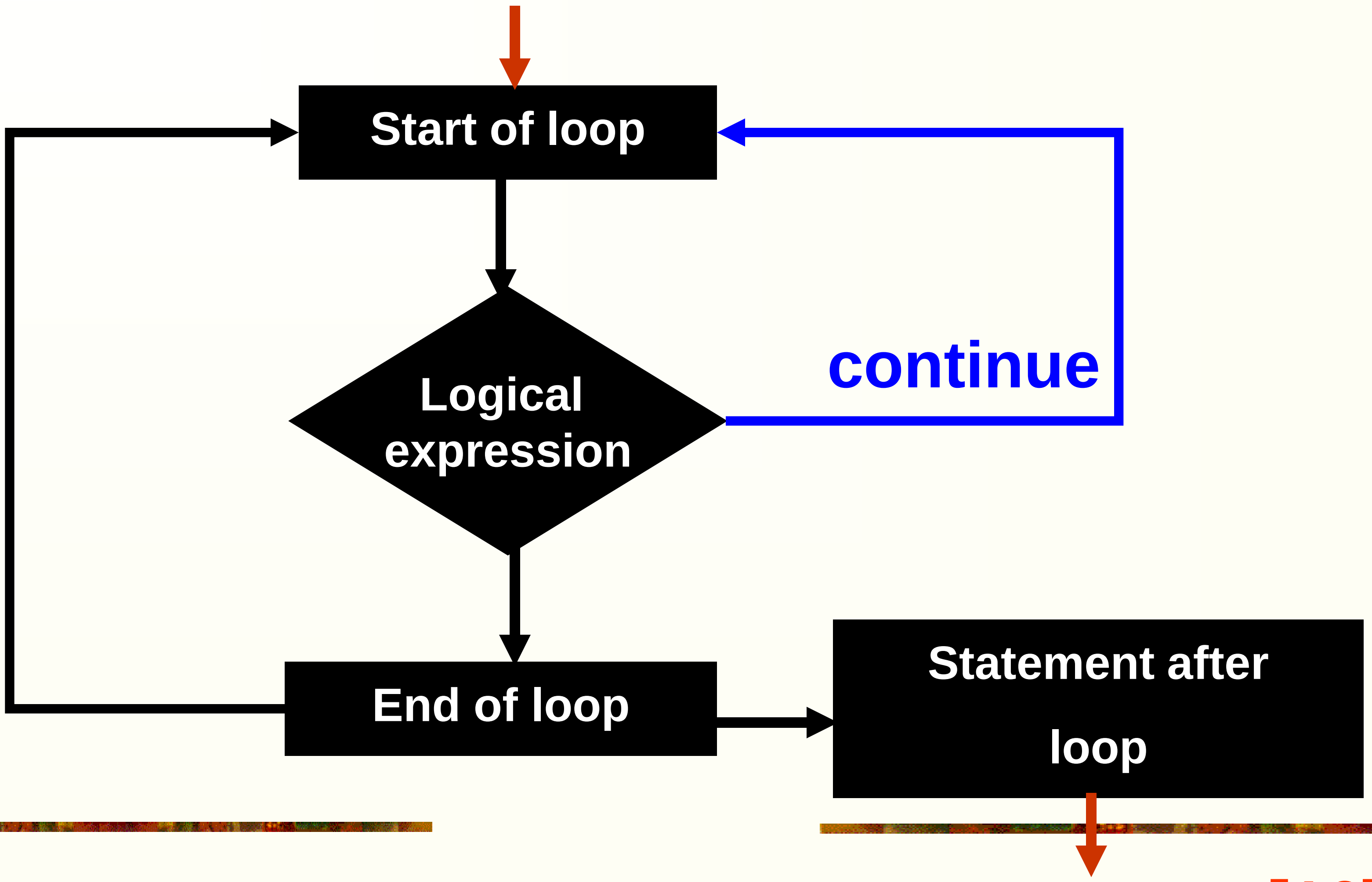
```
//break demo
#include <stdio.h>
main() {
   char userAns = ‘Y’;

   do {
    printf(“There is NO break today”);
    break;
    printf(“display message again (Y/N)?”);
   } while (userAns == ‘Y’);

    printf(“That is all for now”); return 0;
}
```

This program always produces the first and last `printf` statements

continue Statement



`continue` Statement

- Forces the computer to perform another iteration of a loop using the format:

```
continue;
```

- You use the **continue** statement (go back to top of loop to get another value) when data in the body of the loop is bad, out of bounds, unexpected,…

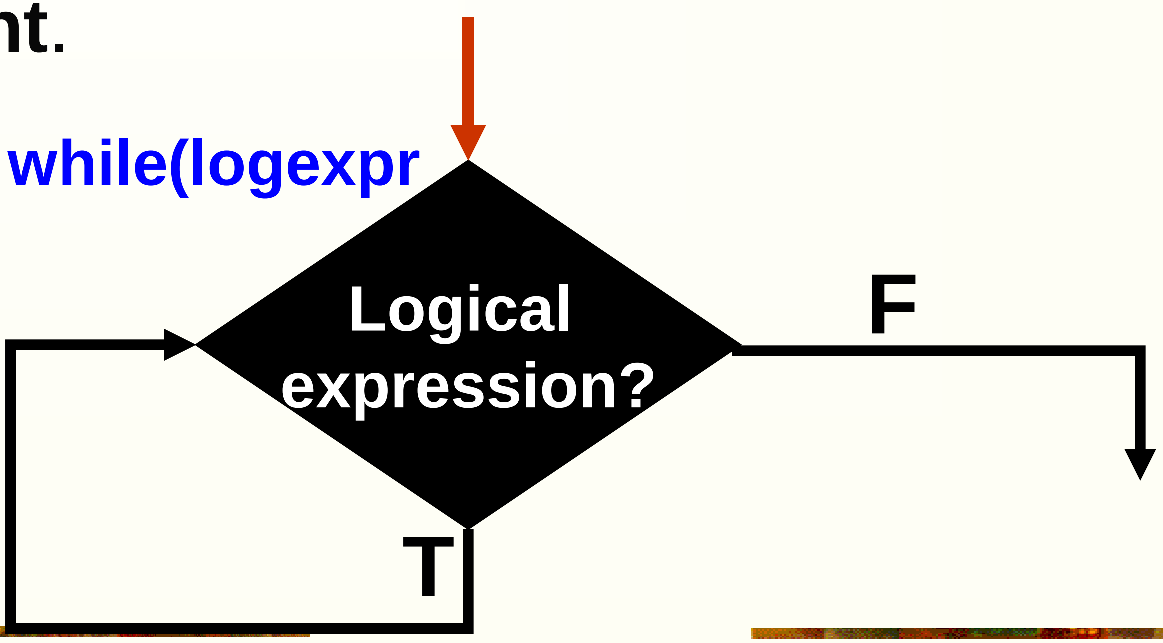
Example

Program demonstrating use of continue in a loop.

```
#include <stdio.h>
int main()
{   int k;
   for (k = 0; k < 5; k++)
      {
      printf("\nFirst");
      if(k%3 == 2) continue;
      printf("\nSecond");
      if(k%3 == 1) continue;
      printf("\nThird");
      }
    return 0;
}
```

Null Statement

- In some cases, we may have a situation where we have a loop that does not have a body. In such cases, the loop is said to have a **null statement**.



Example

```
char name[80];
   int k=0;
   printf("\nEnter your full name :\n");
   while((name[k++]=getchar())!='\n')
       ; /* null statement */
   name[k-1] = '\0';
   printf("\n[%s]",name);
```

Null Statement

Intentionally

Common Bug

```
while (logexpr);
        statement;
```

```
for (.;.;.);
        statement;
```

?????????

Infinite Loops

- When the logical expression used in a while , do-while or for loop remains TRUE forever, the loop is repeated infinitely (until a hardware interrupt is used or the computer is shut off).
- These can be serious bugs in programs.

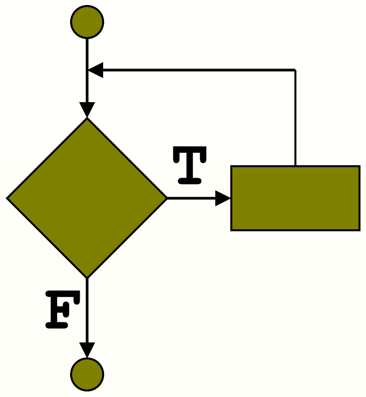
```
while (1)

for(;1;)
```

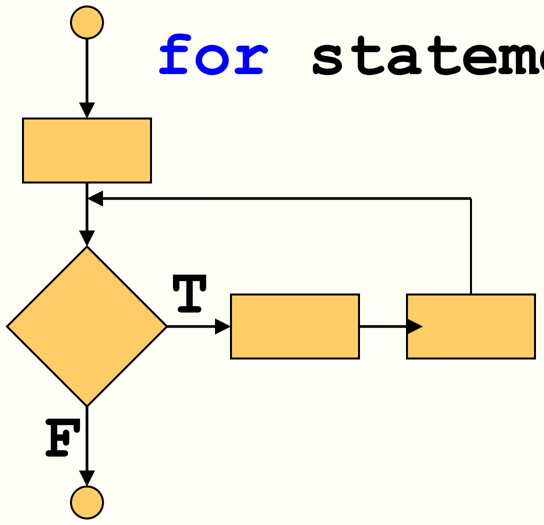
the preferred way to implement an infinite loop

Repetition Structures

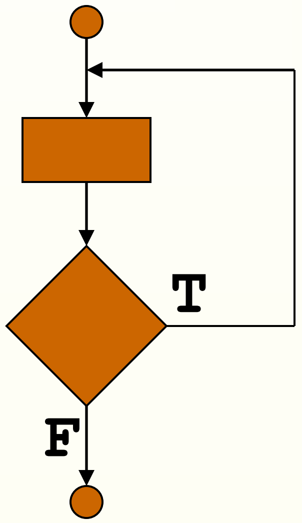
`while` statement



`for` statement



`do…while` statement



Points To Remember

- The while, do-while and for loops are the three iterative constructs in C.
- The break and continue keywords can be used to modify the behavior of these loops.
- We must take care to avoid infinite looping in our programs.
- Infinite loops can be used as valid programming constructs.

THANK YOU