Control Flow of C Program (selection and looping)

A. Sahu and P. Mitra

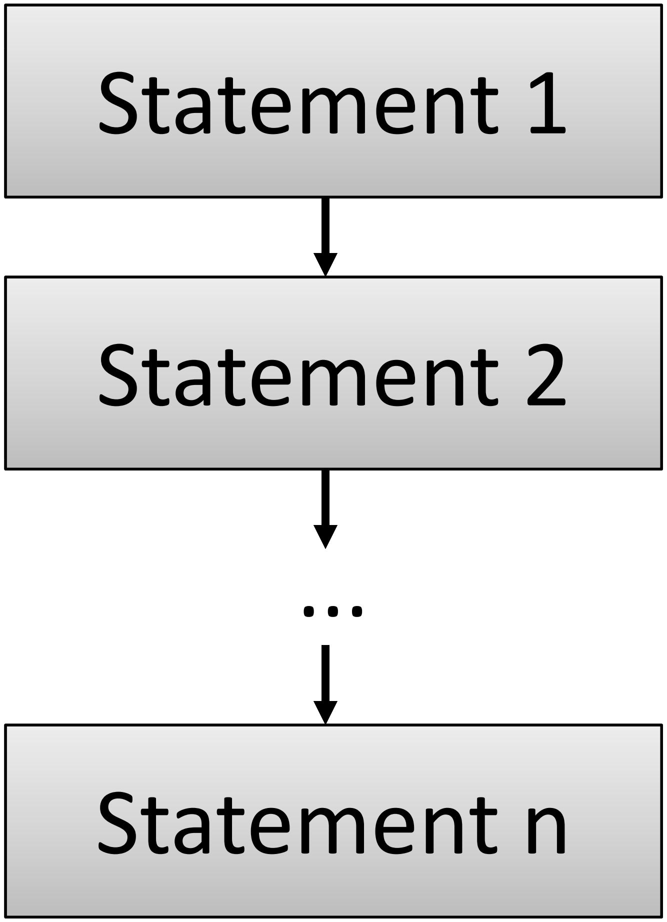
Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Structured Programming

- All programs can be written in terms of only three control structures
  - **Sequence, selection and repetition**
- The **sequence** structure
  - Unless otherwise directed, the statements are executed in the order in which they are written.
- **The selection structure**
  - Used to choose among alternative courses of action.
- **The repetition structur**e
  - Allows an action to be repeated while some condition remains true.

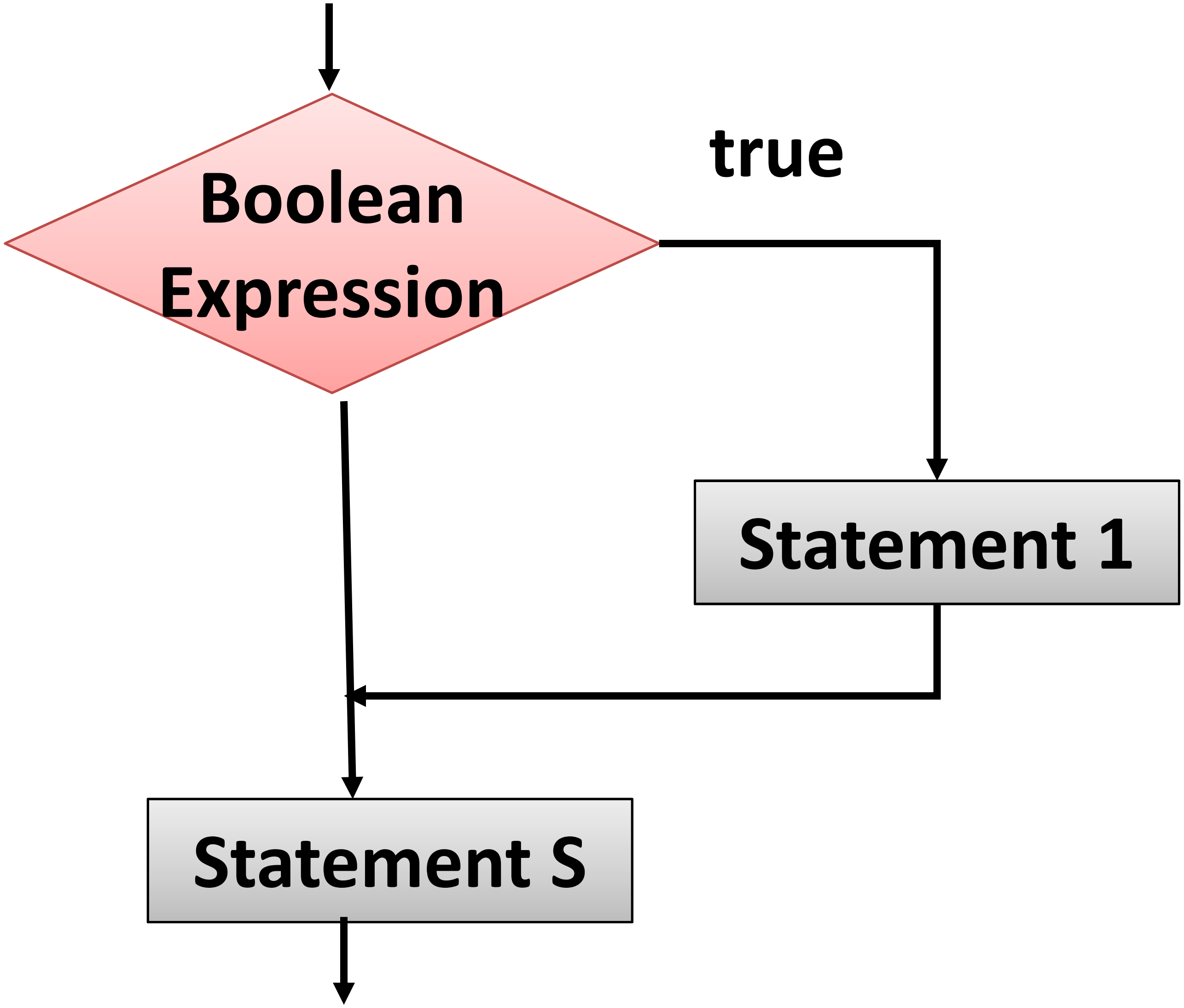
Sequential Execution



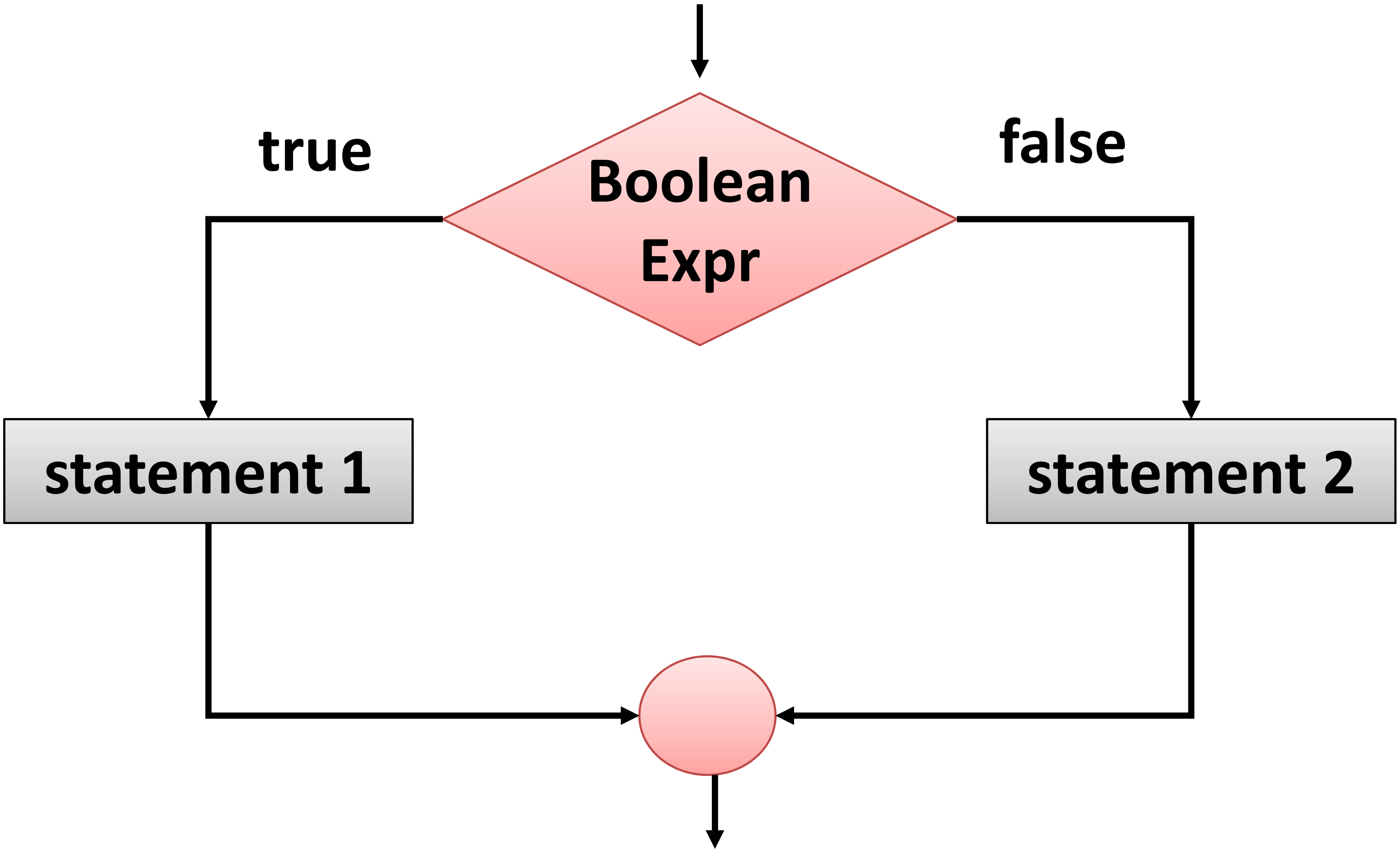
Compute the resonant frequency of an RLC circuit

```c
#include <stdio.h>
#include <math.h>
int main() {
  double l, c, omega, f;
  printf("Enter inductance in mH: ");      //S1
  scanf("%lf", &l);                        //S2
  printf("Enter capacitance in microF: ");  //S4
  scanf("%lf", &c);                        //S5
  omega = 1.0/sqrt((l.0/1000)*(c/1000000)); //S6
  f = omega / (2 * M_PI);                  //S7
  printf("Resonant freq: %.2f\n", f);      //S8
  return 0;                                //S9
}
```

Selective Execution : Flow chart (only if)



Selective Execution : Flow chart



Selection: the if-else statement

```
if  ( condition ){
     statement(s)/*if clause */
}
else {
     statement(s) /*else clause */
}
```

Nesting of if-else Statements

```
if ( $condition_1$ )
{
     statement(s)
}
else if ( $condition_2$ )
{
    statement(s)
}
. . . /* more else clauses may be here */
else
{
    statement(s)  /* the default case */
}
```

Bad Example : 2 if 1 else

```
 if ( n > 0 )
    if ( a > b  )
       z=a;
else
     z=b;
```

```
if ( n > 0 )
  if ( a > b )
     z=a;
  else
     z=b;
```

```
if ( n > 0 )
{
  if (a> b)
       z=a;
 }
 else
     z=b;
```

Indentation will not ensure result :

else match with closest if

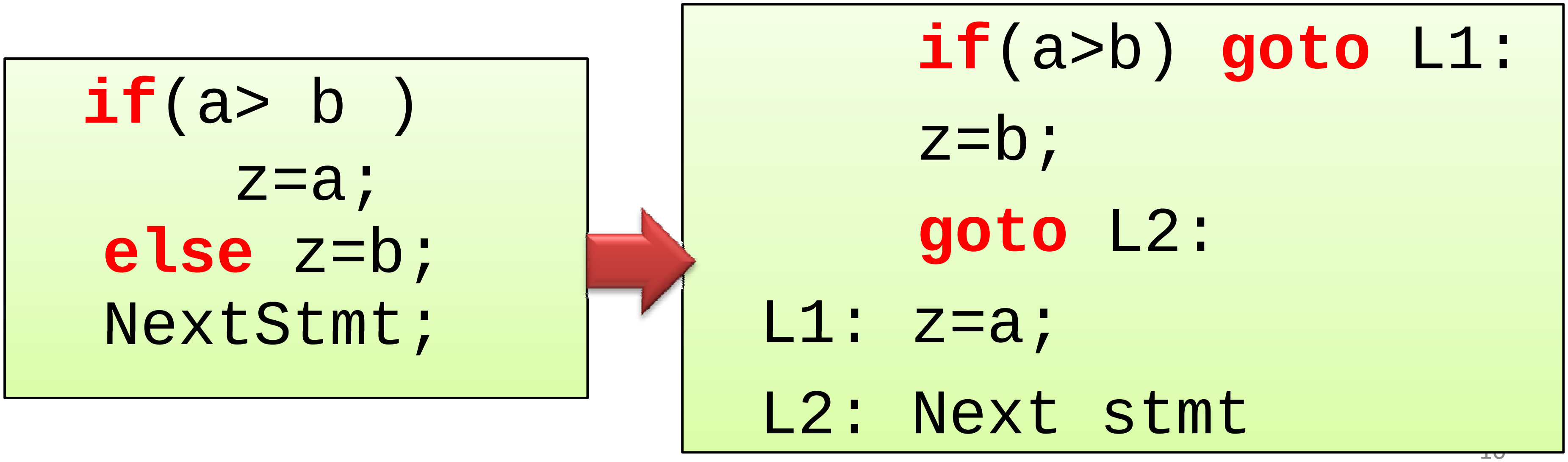
Code of Red box behaves like Code of Green box

In Assembly language: No if-else

- Assembly language
  - No support for [if else, *No for loop, No while loop]*
  - All higher construct get implemented

    using **if** and **goto** statement

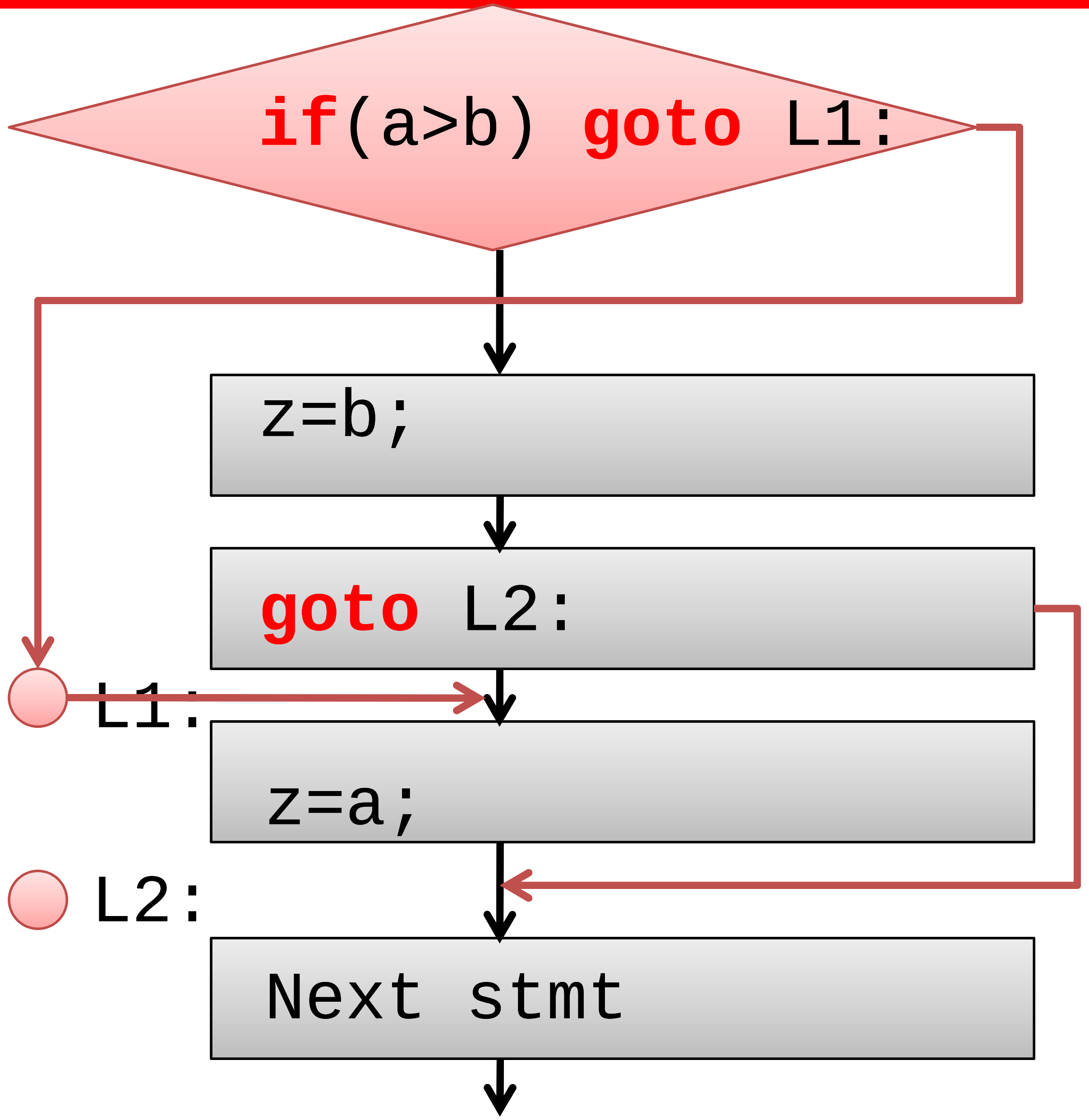
    goto statement uses **Label**
- **If else** get converted to **if goto**



```
  if(a> b )
       z=a;
   else z=b;
   NextStmt;
```

```
        if(a>b) goto L1:
        z=b;
        goto L2:
   L1: z=a;
   L2: Next stmt
```

In Assembly language: No if-else



Multi-way if else : switch case

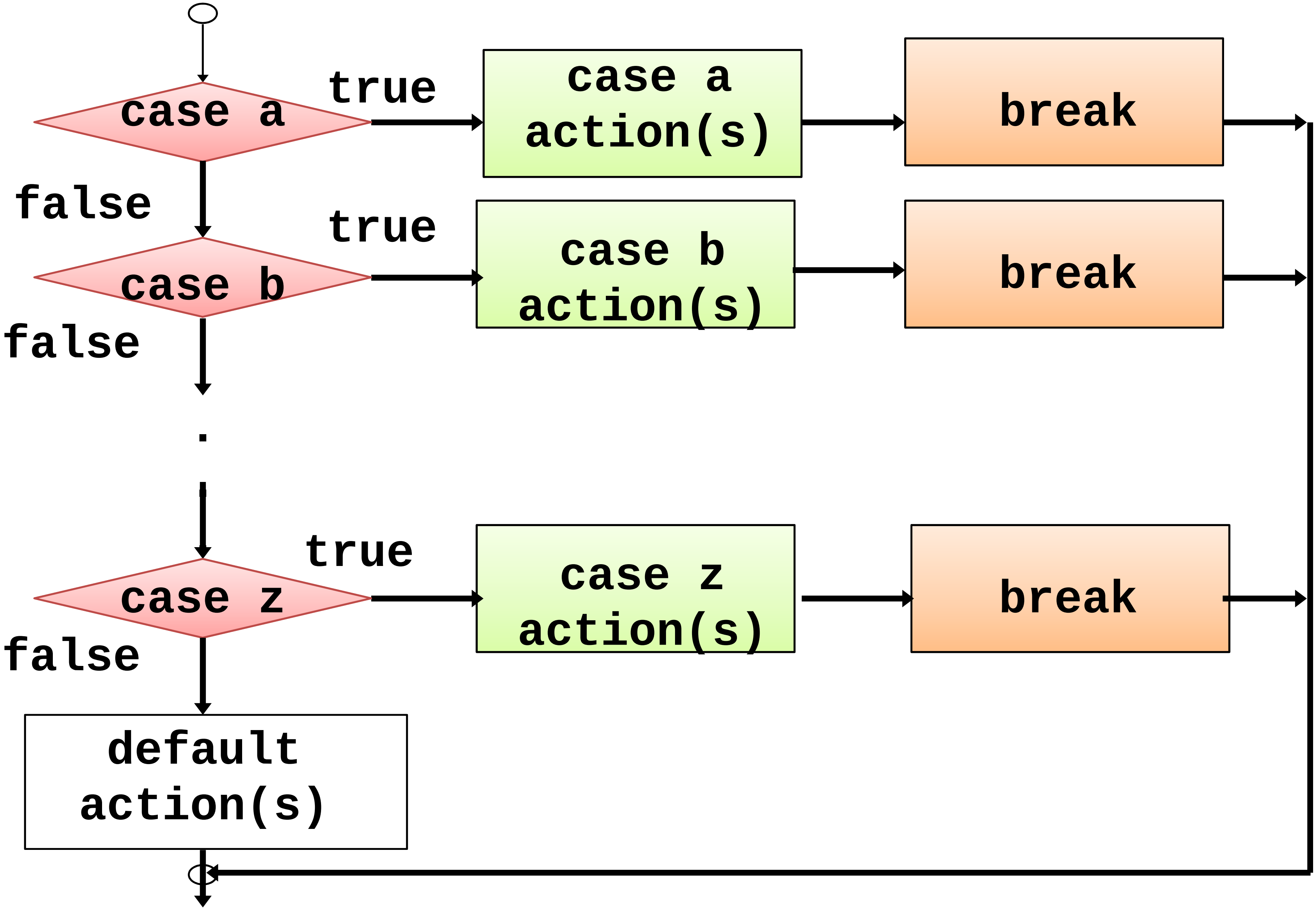
- If-else : two way, if part and else part
- To make it multi-way: nested if-else
  - Confusing, lengthy
- C language provide
  - Switch case
  - Multi way selection
  - Range multi-way selection

The switch Multiple-Selection Structure

- **`switch`**
  - Useful when expression is tested for multiple values
  - Consists of a series of **`case`** labels and an optional **`default`** case
  - **`break`** is (almost always) necessary

```
switch (<expression>){
  case <Value1> :
     <Action/Stmts for Value1>; break;
  case <Value2> :
     <Action/Stmts for Value2>; break;
  . . .
  default: <Action/Stmts for DefaultValue>;
          break;
}
```

Flowchart of Switch Statement



Multiway Switch Selection example

```
int main(){//simple calculator
 int a=50,b=10, R;
 char choice;
 printf(“Enter choice”);
 scanf(“%c”,&choice);

 switch (choice){
  case ‘a’ : R=a+b; printf(“R=%d”,R); break;
  case ‘s’ : R=a-b; printf(“R=%d”,R); break;
  case ‘m’ : R=a*b; printf(“R=%d”,R); break;
  case ‘d’ : R=a/b; printf(“R=%d”,R); break;
  default  : printf(“Wrong choice”) ; break;
  }
 return 0;
}
```

Multiway Switch Selection example

```
int main(){//simple calculator
 int a=50,b=10, R;
 char choice;
 printf(“Enter choice”);
 scanf(“%c”,&choice);

 switch (choice){
  case ‘a’ : R=a+b; printf(“R=%d”,R); break;
  case ‘s’ : R=a-b; printf(“R=%d”,R); break;
  case ‘m’ : R=a*b; printf(“R=%d”,R); break;
  case ‘d’ : R=a/b; printf(“R=%d”,R); break;
  default  : printf(“Wrong choice”) ; break;
  }
 return 0;
}
```

Multiway Switch Selection example

```
switch (choice){
 case ‘A’ : // no break, work for both A & a
            // next statement automatically
            // get executed
 case ‘a’ : R=a+b; printf(“R=%d”,R); break;
 case ‘S’ :
 case ‘s’ : R=a-b; printf(“R=%d”,R); break;
 case ‘M’ :
 case ‘m’ : R=a*b; printf(“R=%d”,R); break;
 case ‘D’ :
 case ‘d’ : R=a/b; printf(“R=%d”,R); break;
 default  : printf(“Wrong choice”) ; break;
 }
```

Range Multiway Switch Selection example

```
int x;
scanf(“%d”,&x);
switch (x){
 case 1 ... 20:// 1 space three dots space 20
    printf(“You entered >=1 and <=20”);
    break;
 case 21 ... 30 :
     printf(“You entered >=21 and <=30”);
    break;
 default  :
    printf(“You entered < 1 and >31”) ;
    break;
 }
```

```
Syntax = case <low_range> ... <high_range>  :
```

Loops and Repetition

- Loops in programs allow us to repeat blocks of code
- Useful for:
  - Counting
  - Repetitive activities
  - Programs that never end
- **Because of looping feature of computer**
  - **We also name “Computer” as “Machine” (which can do repetitive mechanical work for us)**

Three Types of Loops/Repetition in C

- **while**
  - top-tested loop (pretest)
- **for**
  - counting loop
  - forever-sentinel
- **do**
  - bottom-tested loop (posttest)

The while loop

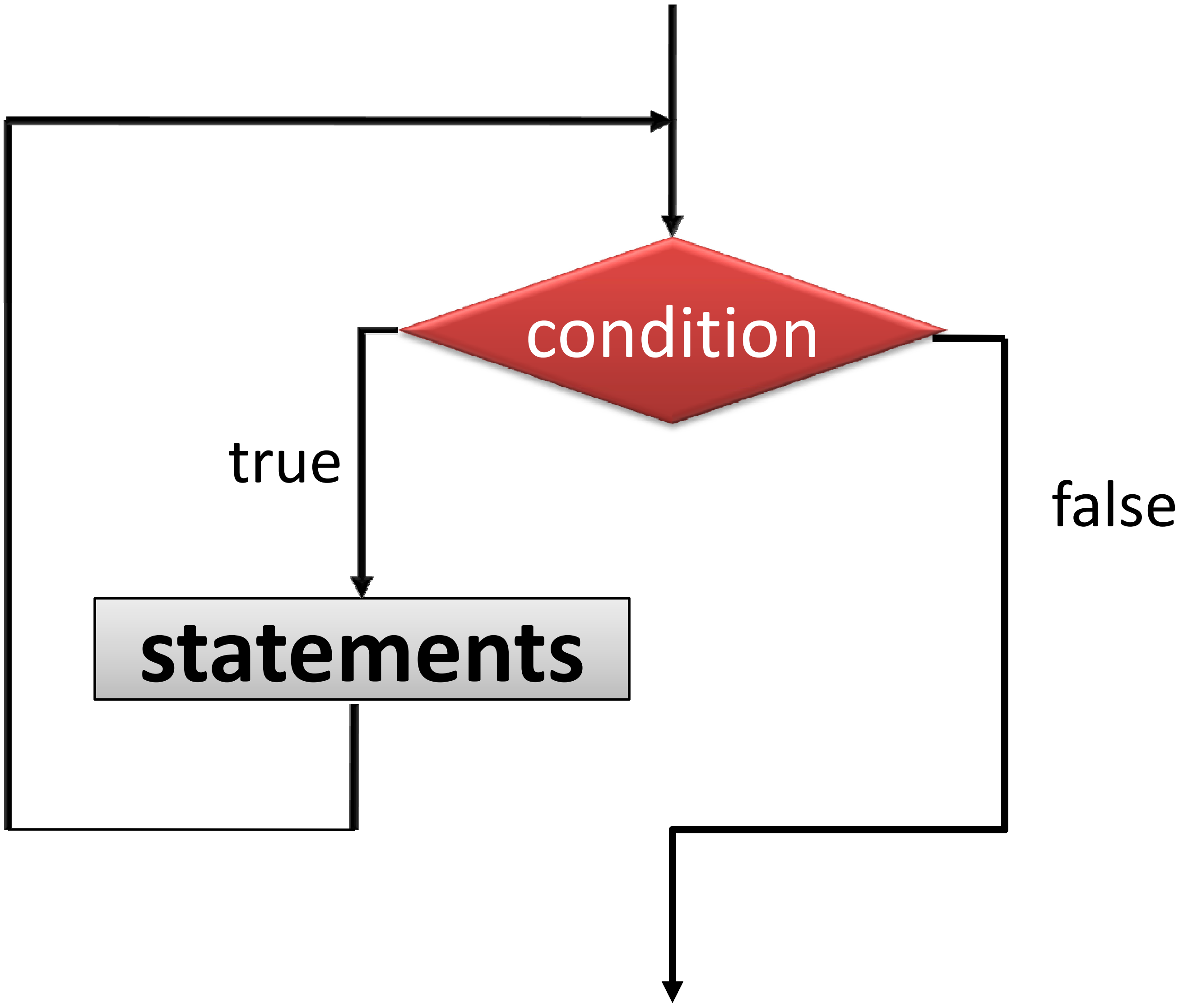
Top-tested loop (pre-test)

```
while (condition)
   statement;
```

Note that, as in IF selection, only one statement is executed. You need a block to repeat more than one statement (using { })

```
while (condition){
   statements;
}
```

while(condition)statement;



Similar to the if statement

- Check the **Boolean** condition
- If true, execute the statement/block

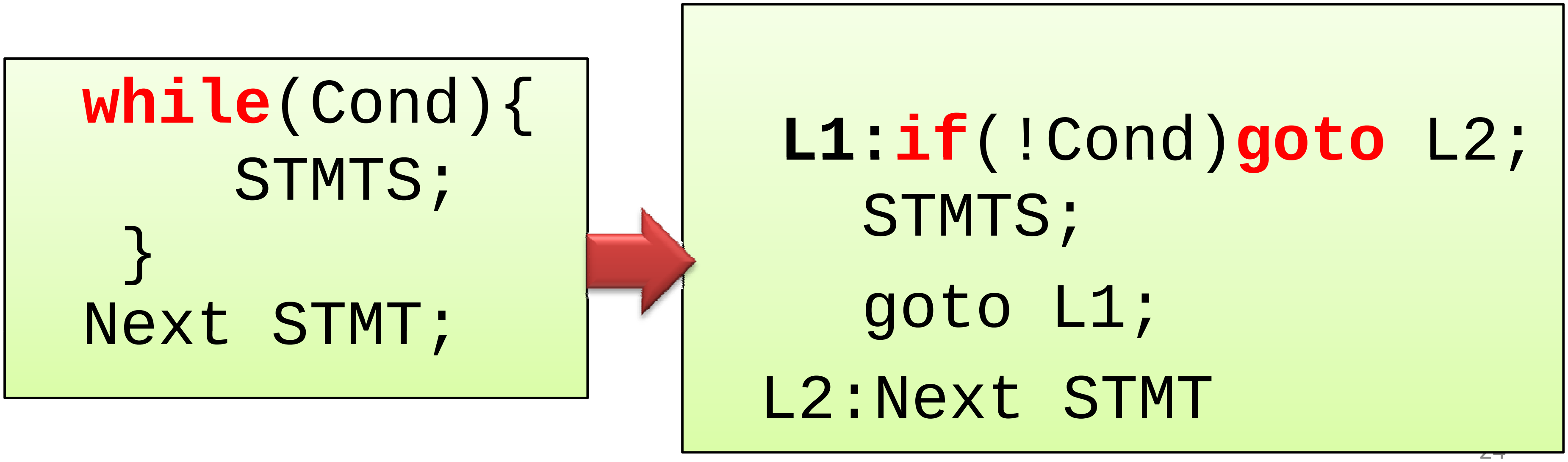
Repeat the above until the **Boolean** is false

In Assembly language: No while loop

- Assembly language
  - No support for [*while loop]*
  - All higher construct get implemented

    using **if** and **goto** statement

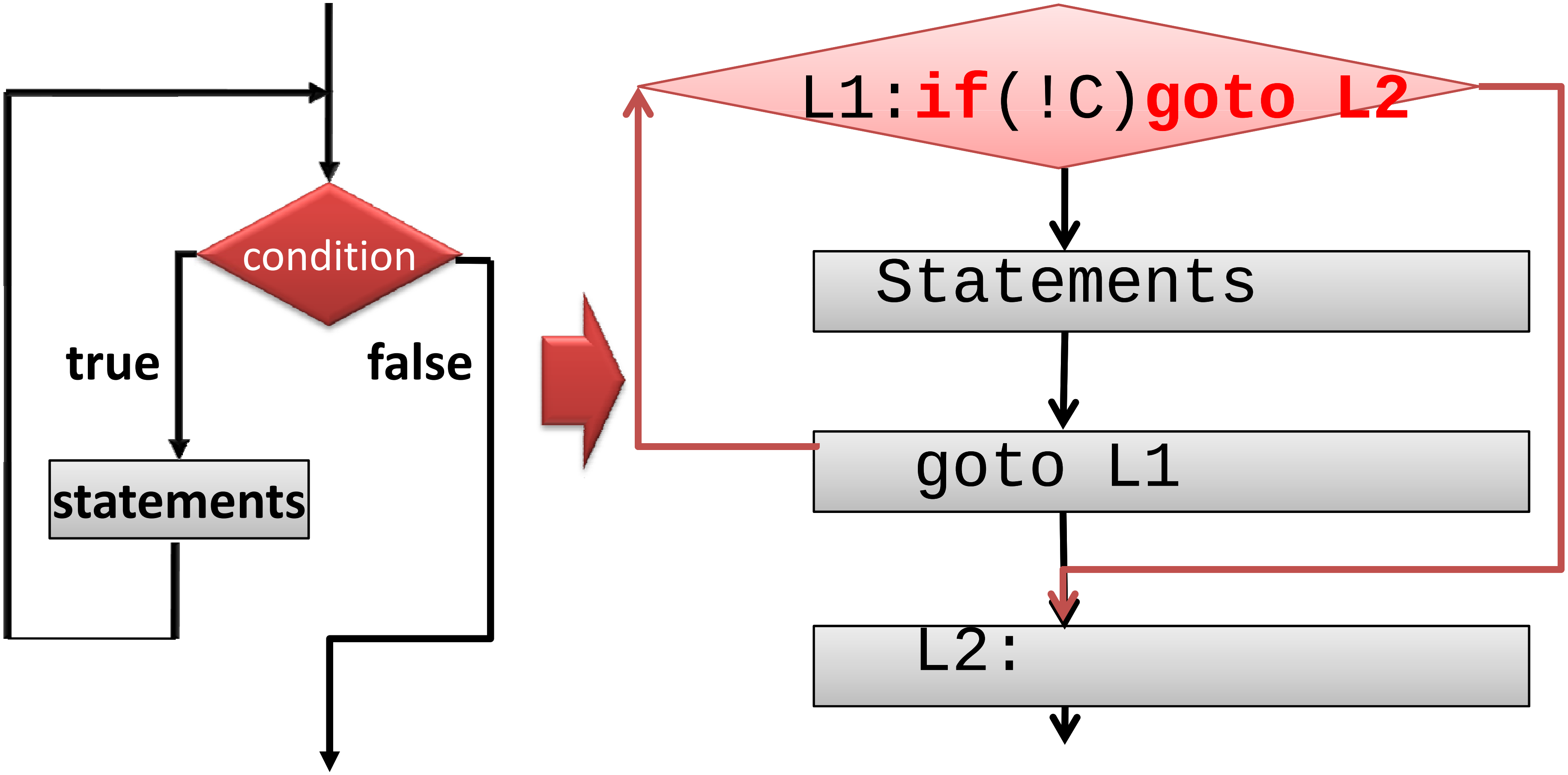
    goto statement uses **Label**
- **while** get converted to **if goto**



```
while(Cond){
    STMTS;
 }
Next STMT;
```

```
 L1:if(!Cond)goto L2;
   STMTS;
   goto L1;
L2:Next STMT
```

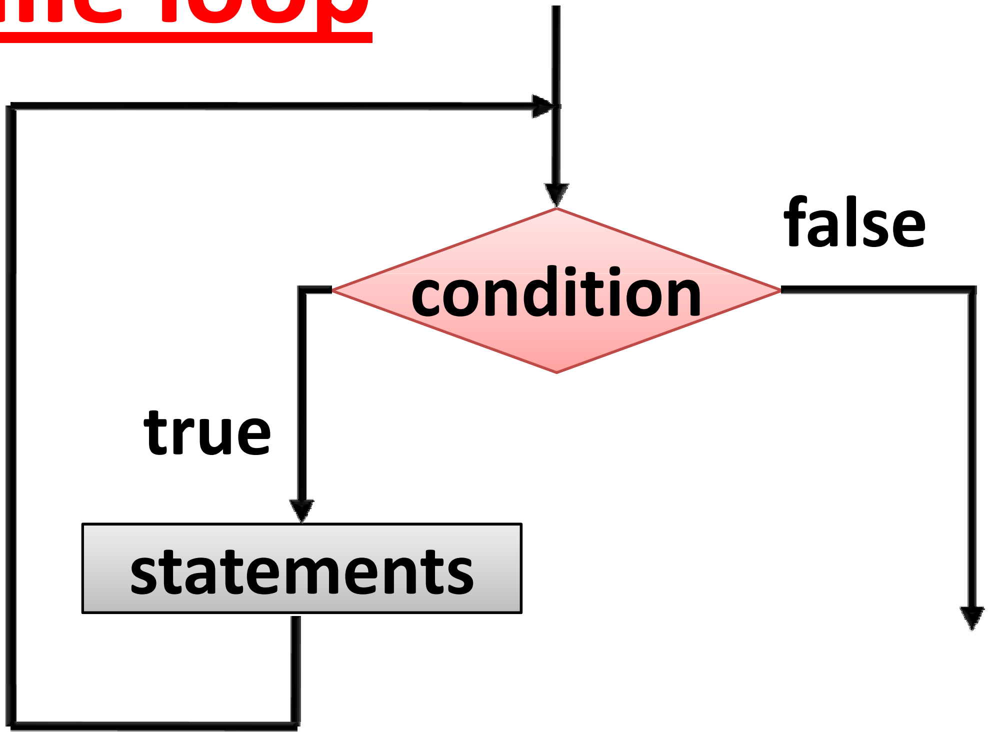
While statement using goto



While loop

```
while(condition)
    statement;

while(condition){
    statement1;
    statement2;
}
```



false

condition

true

statements

```
int i = 10;
while(i > 0) {
    printf("i=%d\n", i);
    i = i - 1;
}
```

Forever loops and never loops

- Because the conditional can be
  - “always true” : you can get a loop that runs **forever**
  - or “always false”, you can get a loop never runs at all.

```
int count=0;
while(count !=0)
  printf(“Hi .. \n”);// never prints

while (count=1)//insidious error!!!
  count = 0;
```

What is wrong with these statements?

How to count using while

1. First, outside the loop, initialize the counter variable
2. Test for the counter’s value in the Boolean
3. Do the body of the loop
4. Last thing in the body should change the value of the counter!

```
1.      i = 1;
2.      while(i <= 10)  {
3.         printf("i=%d\n", i);
4.         i = i + 1;
        }
```

The for loop

- The **while** loop is pretty general.
  - Anything that can be done **using repetition** can be done **with a while loop**
- Because counting is so common
  - There is a specialized construct
  - Called : **for loop**.
- **for loop**
  - Makes it easy to set up a counting loop

For loop: Three parts

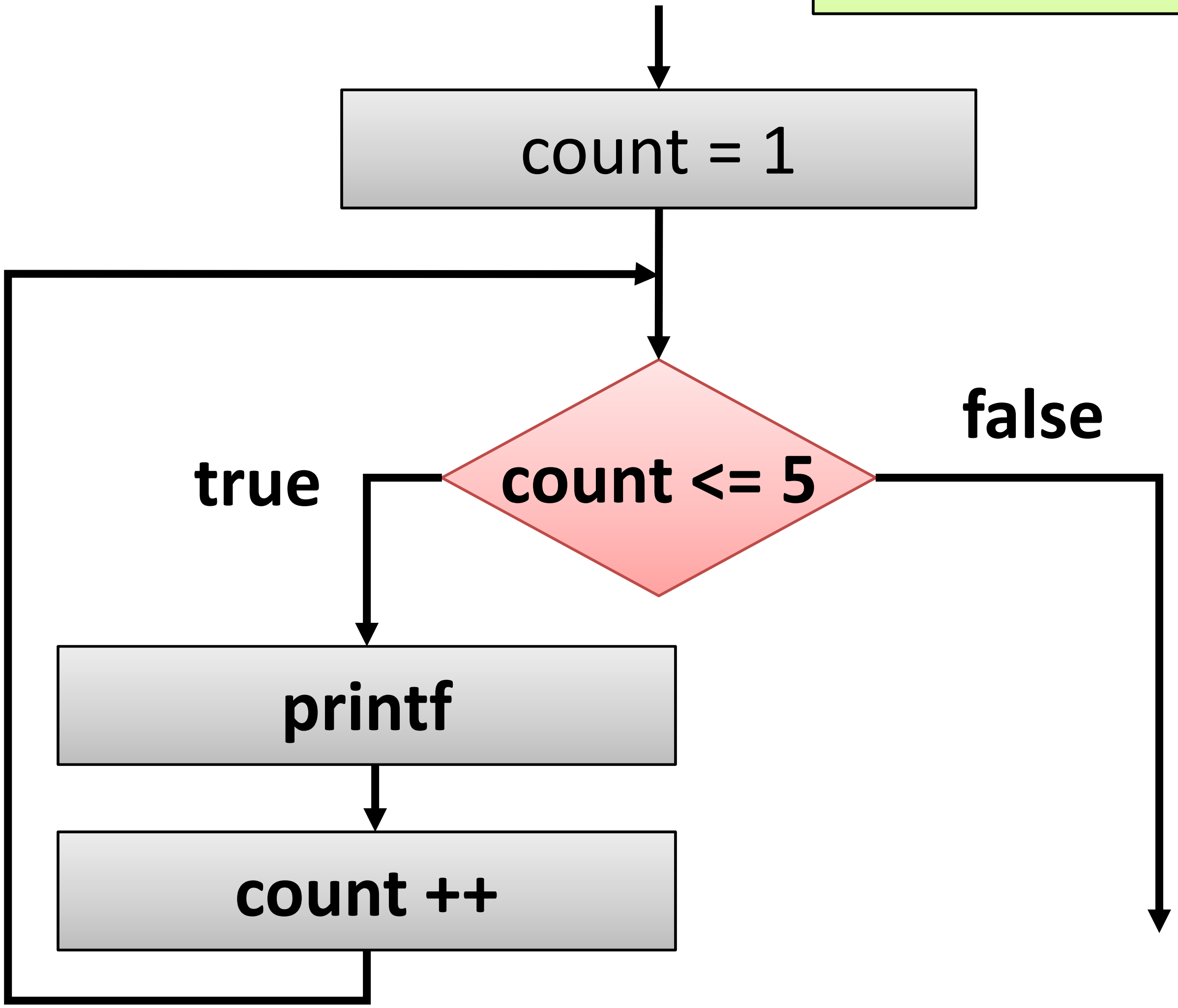
Three parts to a for loop (just like the while):

- Set the initial value for the counter
- Set the condition for the counter
- Set how the counter changes each time through the loop

```
for ( count=1; count<=5; count++ ){
    statement;
}
```

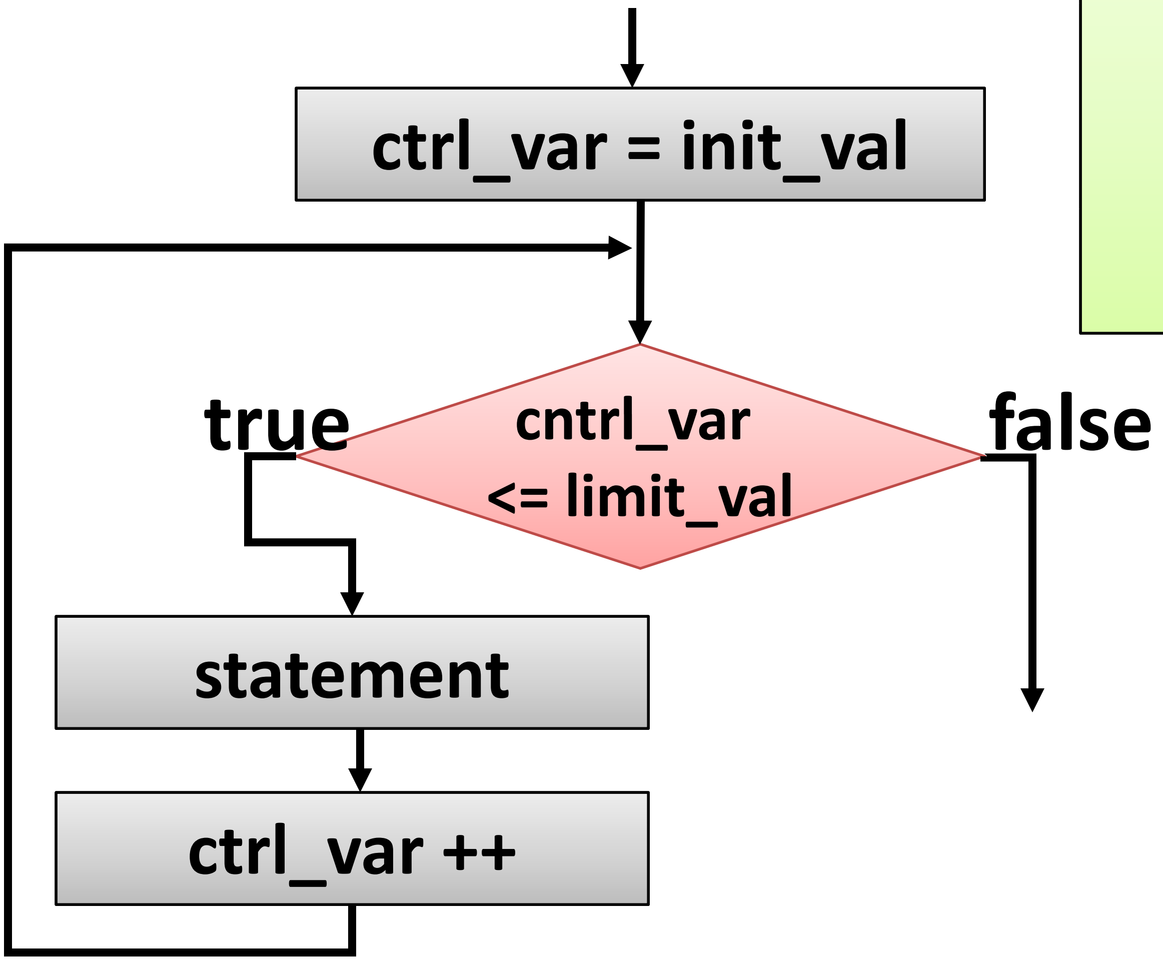
For Loop: Example

```
for(count=1; count<=5; count++)
    printf(“count=%d\n”, count);
```



For loop: Ascending for

```
for ( ctrl_var=init_val;
      ctrl_var <=limit_val;
      ctrl_var++) {
         statement;
     }
```



ctrl_var = init_val

cntrl_var <= limit_val

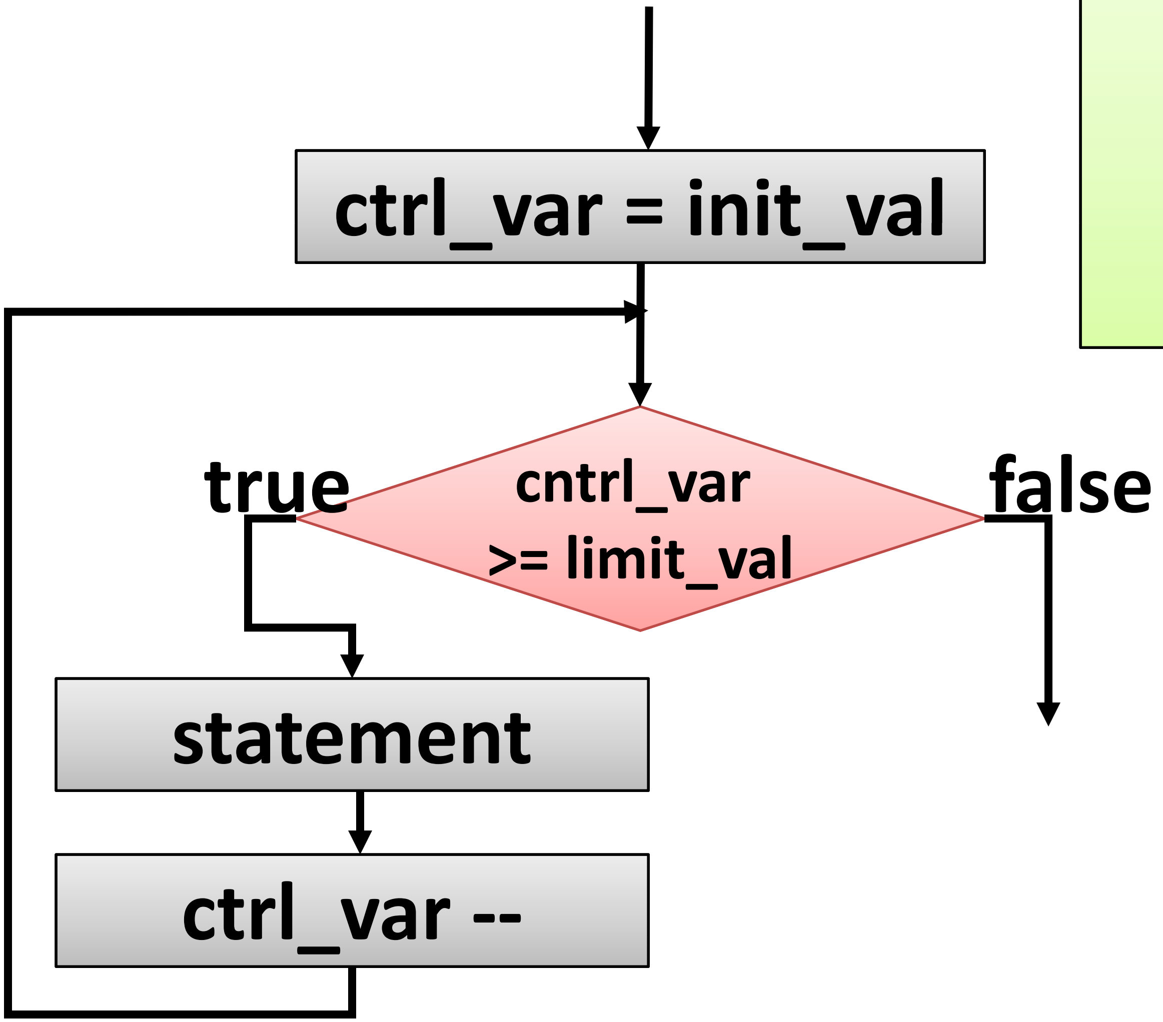
true

false

statement

ctrl_var ++

For Loop : Descending for



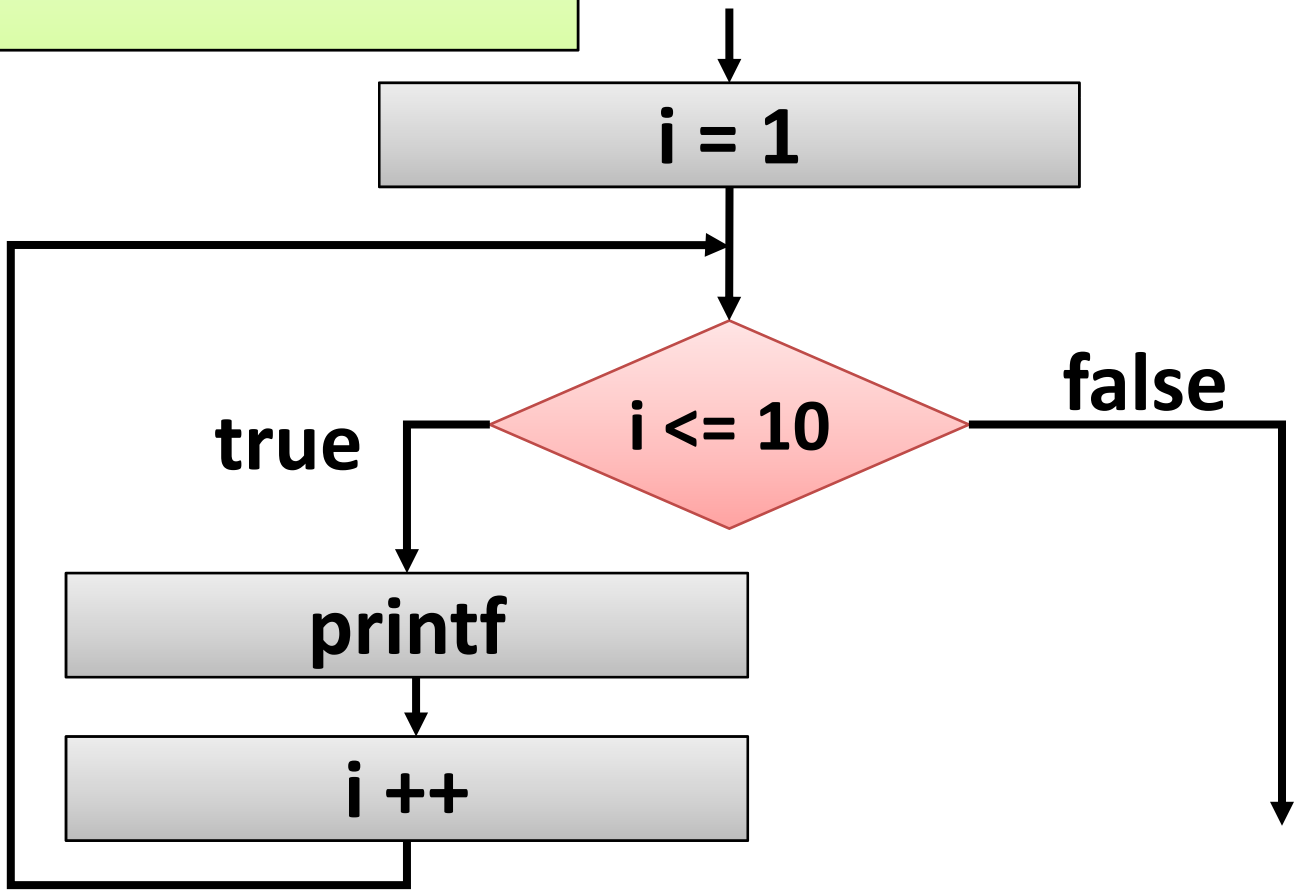
```
for ( ctrl_var=init_val;
      ctrl_var >=limit_val;
      ctrl_var--) {
          statement;
     }
```

Precaution in Coding

- Dangerous to alter within the body of the loop
  - control variable ctrl_var
  - limit_var
- Components of the for statement can be a arbitrary statements
  - e.g. the loop condition may be a function call.

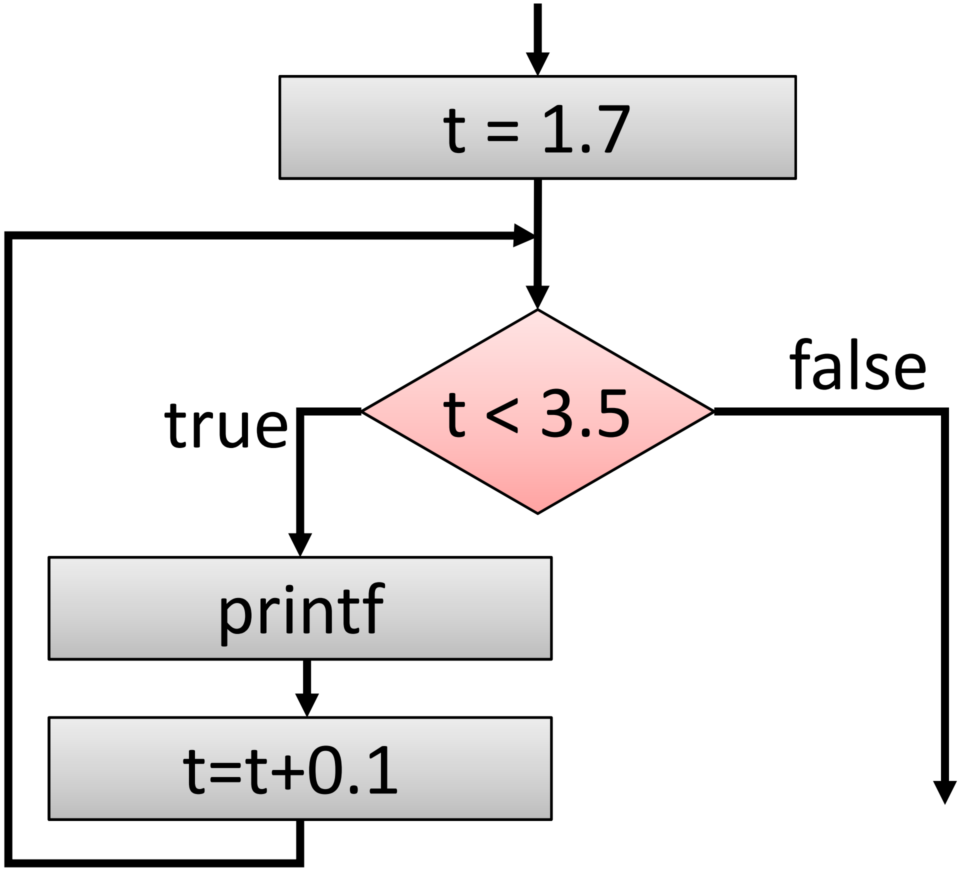
For loop : Examples

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



For loop Examples : Float ctrl variable

```
for(t=1.7; t<3.5; t=t+0.1){
   printf("%f\n", t);
}
```



For Loop: “one off” error

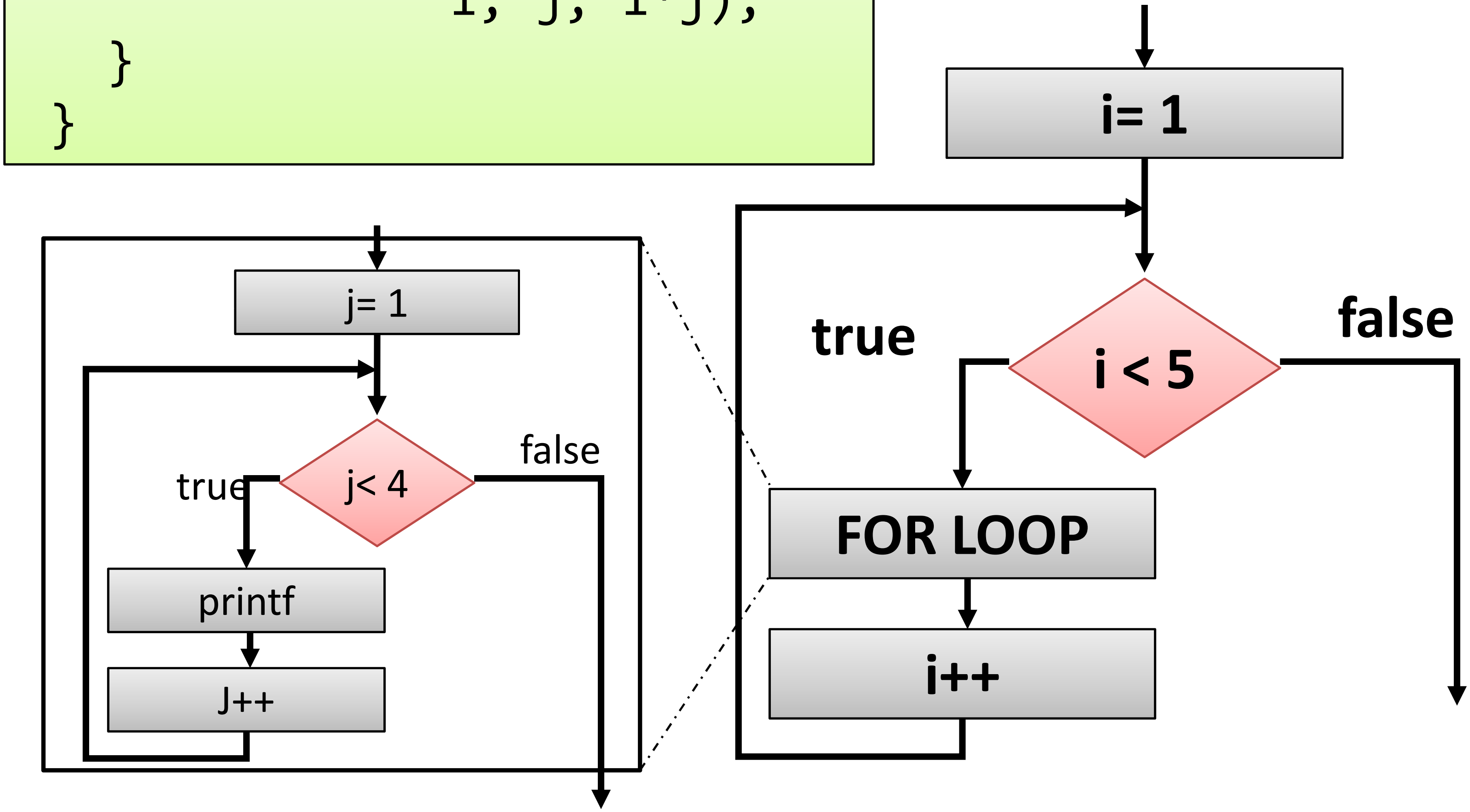
- It is easy to get a for loop to be “one off” of the number you want.
- Be careful of the combination of **init_value** and < vs. <=
  - `for(i=0; i<10;  i++)`
  - `for(i=0; i<=10; i++)`
  - `for(i=1; i<10;  i++)`
  - `for(i=1; i<=10; i++)`

For Loop: “one off” error

- It is easy to get a for loop to be “one off” of the number you want.
- Be careful of the combination of **init_value** and < vs. <=
  - `for(i=0; i<10;  i++)` **10 values: 0 to 9**
  - `for(i=0; i<=10; i++)` **11 values: 0 to 10**
  - `for(i=1; i<10;  i++)` **9 values: 1 to 9**
  - `for(i=1; i<=10; i++)` **10 values: 1 to 10**
- Counting from 0, with <, is a good combination and good for invariants as well
  - **As array indexing in C start with 0 : will be discussed later**

Nested For loop : Examples

```
for(i=1; i<5; i++){
  for(j=1;  j<4;  j++){
    printf("%d * %d = %d\n",
              i, j, i*j);
  }
}
```



Equivalence of top-tested while loop and for loop

- The following loop

```
for(x=init; x<=limit; x++){
    statement_list
}
```

- Is equivalent to

```
x=init;
while (x<=limit) {
    statement_list;
    x++;
}
```

For-ever or infinite loop

- Used for event driven case
- Mostly event break the infinite loop using break statement : **coming out of the loop**

```
 while(1) {
     /* Loop until value is valid */
}
```

```
  for(;;) {
     /* Loop without testing */
}
```

For-ever or infinite loop

- Used for event driven case
- Mostly event break the infinite loop using break statement

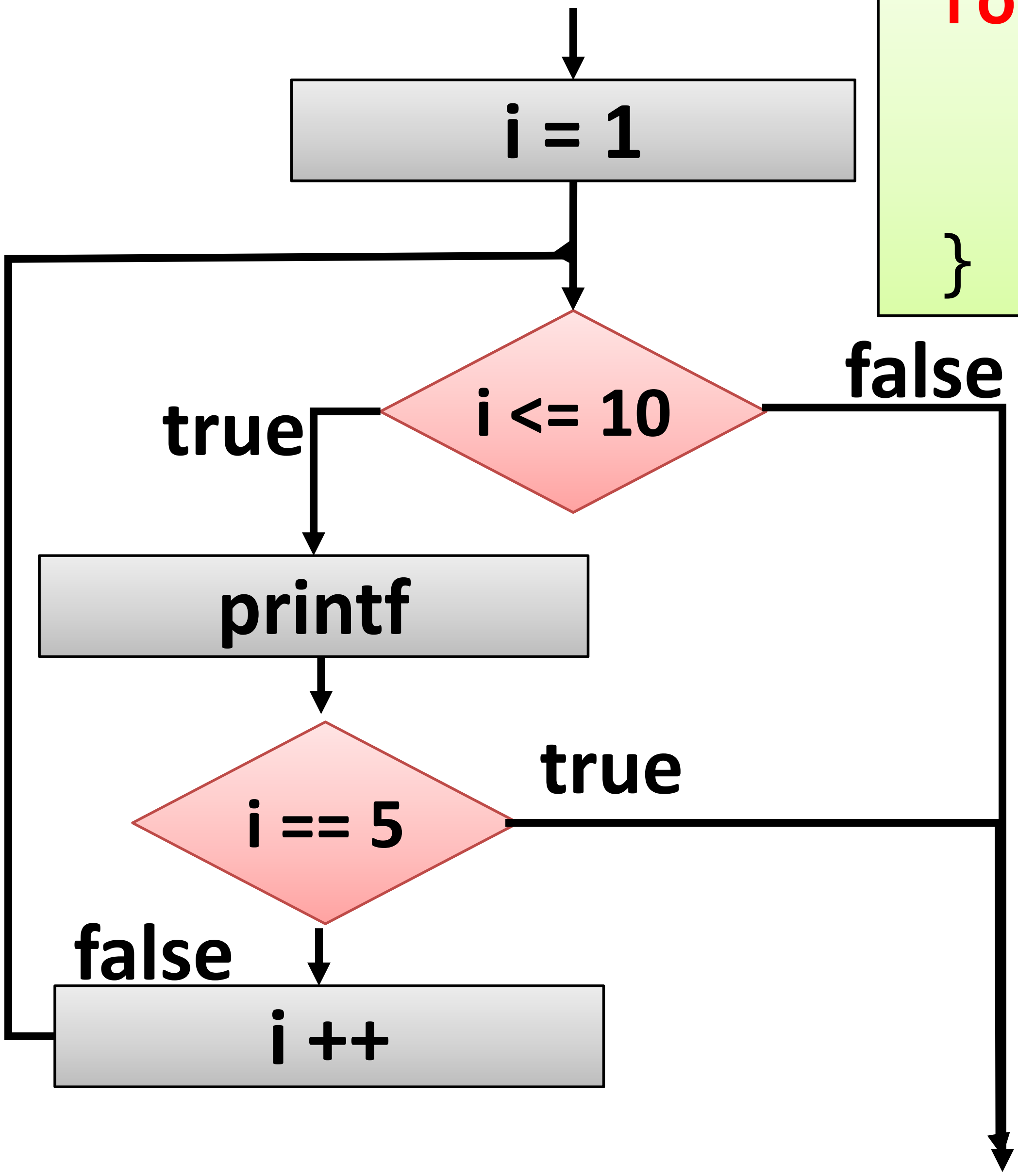
```
while(1) {
    scanf(“%c”,&c);
    if(c==‘e’ || c==‘E’) {
      printf(“\nEntered the required
             character e or E\n”);
      break; // coming out of the loop
      }
    printf(“%c”,c);
}
```

For-ever or infinite loop

- Used for event driven case
- Mostly event break the infinite loop using **break** statement

```
   for(;;) {
      scanf(“%c”,&c);
       if(c==‘e’ || c==‘E’) {
          printf(“\nEntered the required
                character e or E\n”);
          break; // coming out of the loop
          }
     printf(“%c”,c);
}
```

Finite for loop with : break

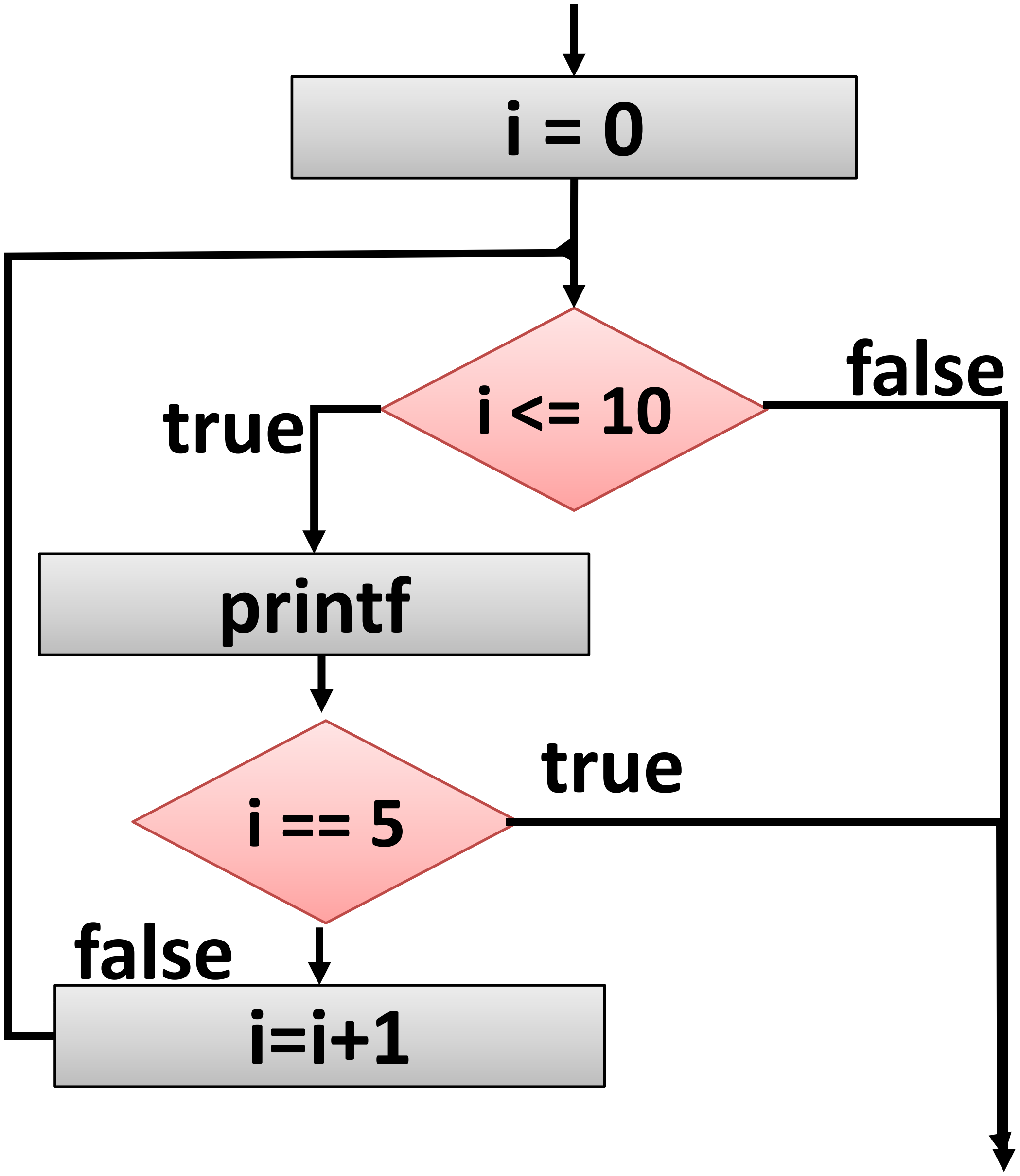


```
for(i=1; i<=10; i++){
  printf("%d, ", i);
  if(i==5) break;
}
```

```
1, 2, 3, 4, 5,
```

// coming out of the loop

Finite while loop with : break

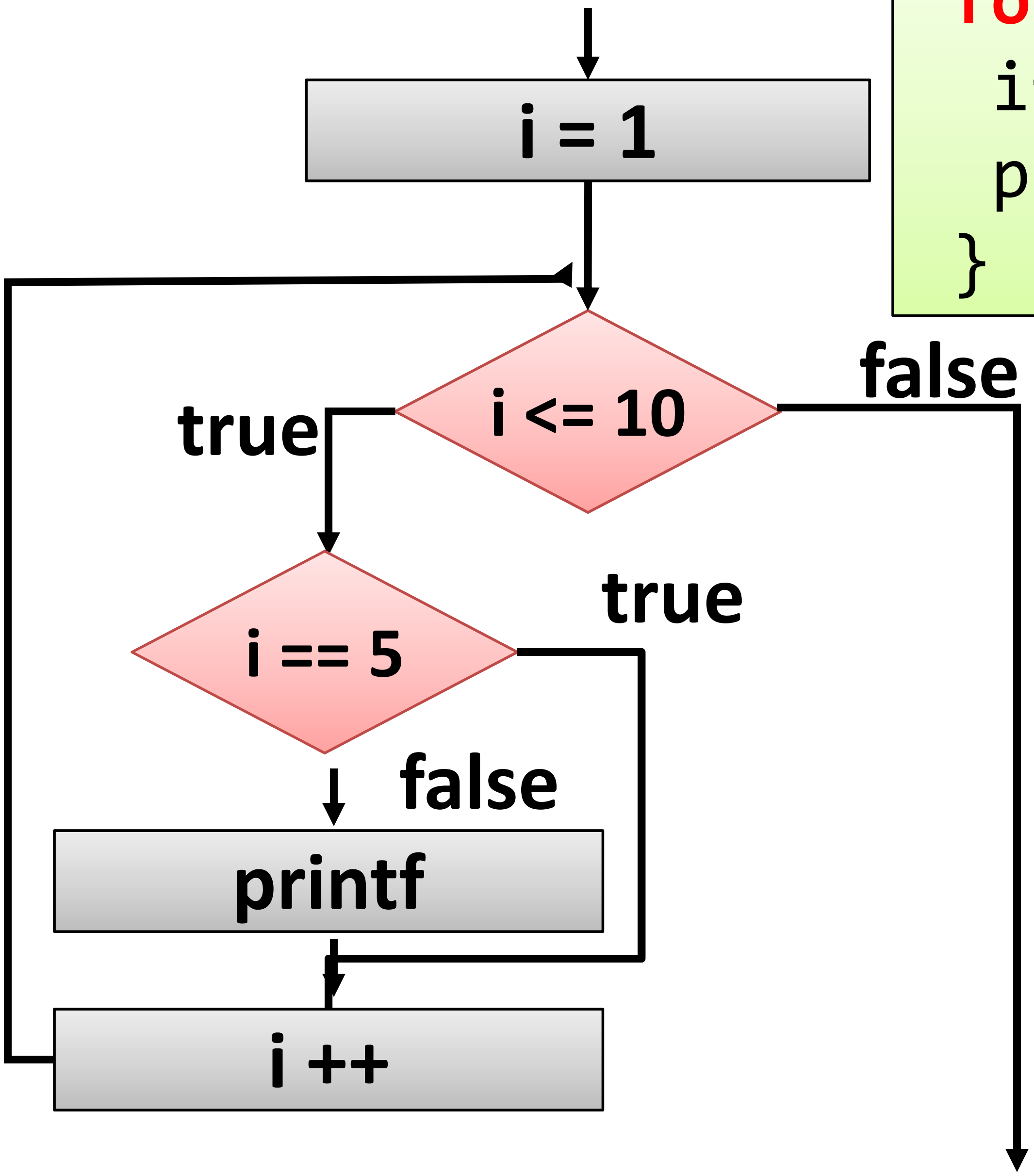


```
i=0;
while(i<=10){
   printf("%d, ", i);
   if(i==5) break;
   i=i+1;
 }
```

```
0, 1, 2, 3, 4, 5,
```

// coming out of the loop

Finite for loop with : continue

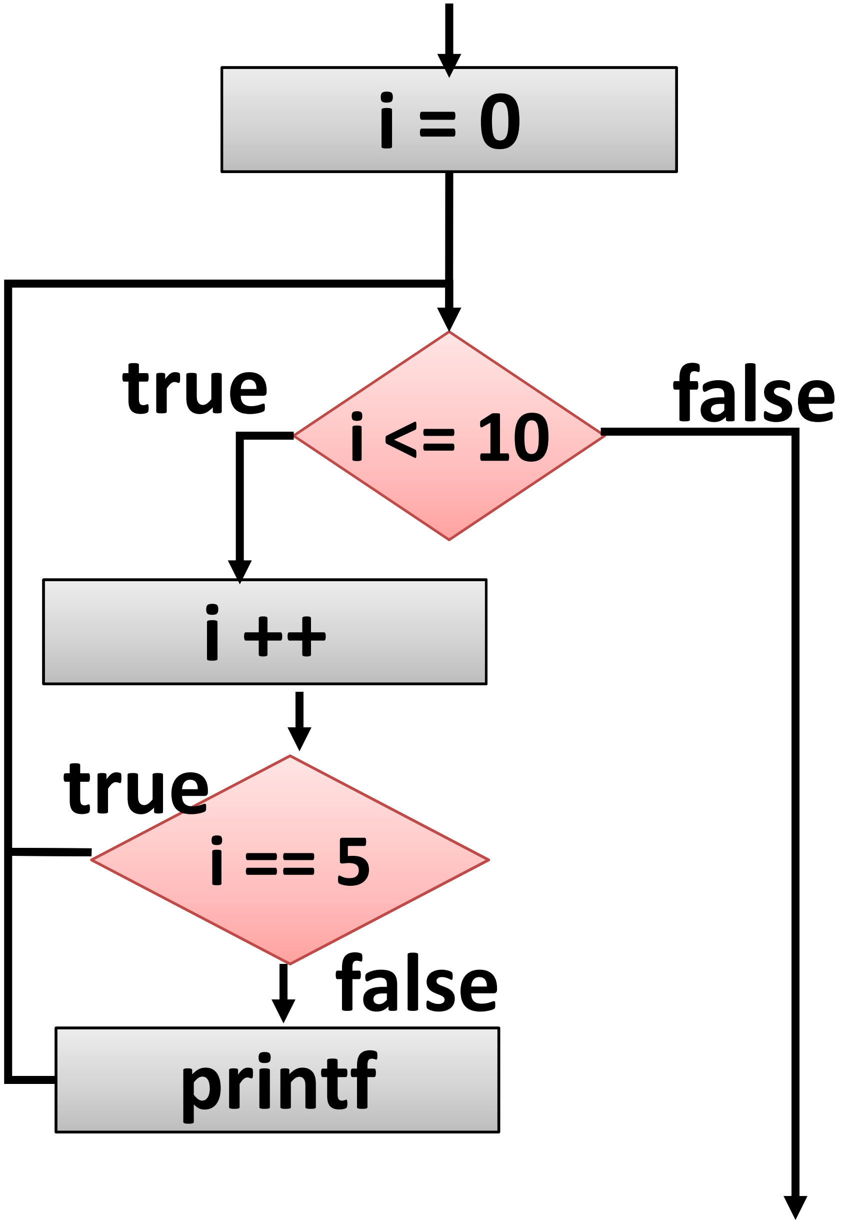


```
for(i=1; i<=10; i++){
  if(i==5) continue;
  printf("%d\n", i);
}
```

0, 1, 2, 3, 4, 6, 7, 8, 9, 10,

// Skip a case if condition satisfied

Finite while loop with : continue



```
 i=0;
While(i<=10){
  i++;
  if(i==5) continue;
  printf("%d, ", i);
 }
```

```
1, 2, 3, 4, 6, 7, 8, 9, 10,
```

// Skip a case if condition satisfied

The do-while loop

- bottom-tested loop (posttest)
- One trip through loop is guaranteed, i.e. statement is executed at least once

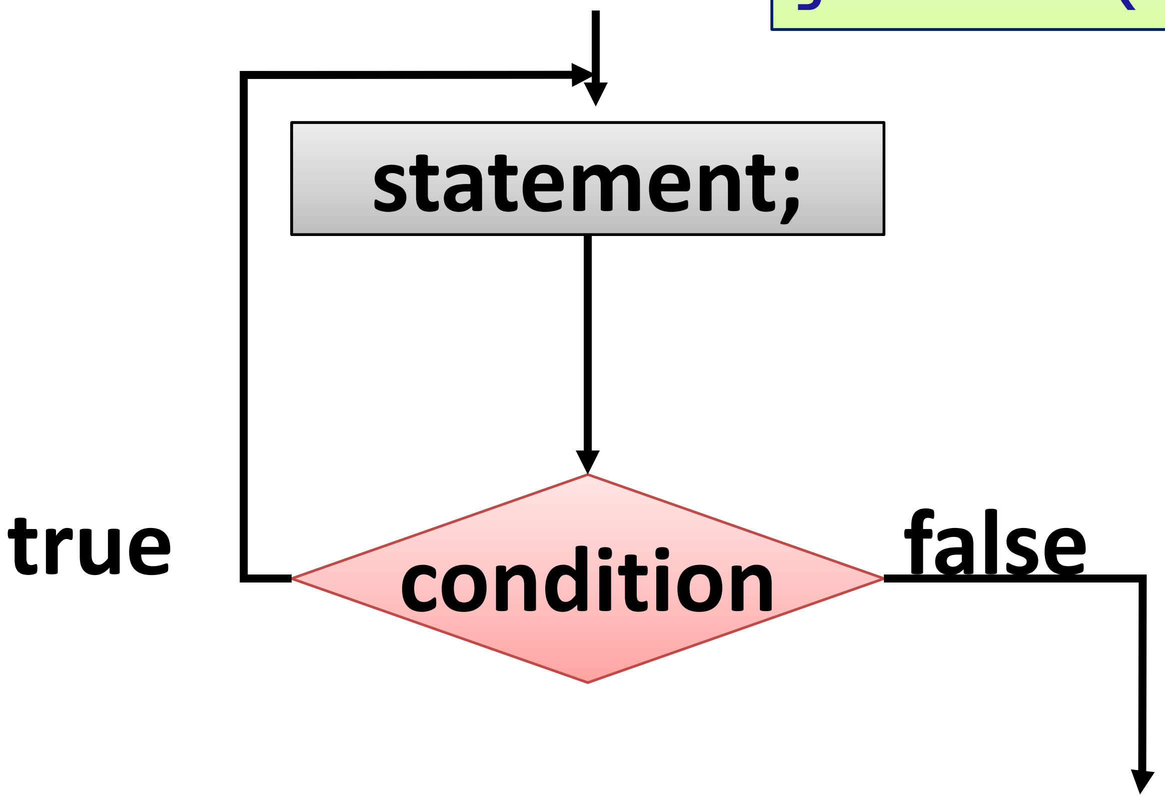
```
do
   statement
while (loop_condition);
```

```
do {
   statement1;
   statement2;
} while (loop_condition);
```

Usually!

do-while loop

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



do loop Examples

```
i = 0;
do {
  i++;
  printf("%d\n", i);
} while(i < 10);
```

```
 do {
   printf("Enter a value>0: ");
   scanf("%lf", &val);
   } while(val <= 0);
```

Bottom-tested Equivalence

- Bottom-tested do loop (posttest)

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

- Similar to bottom-tested forever loop

```
for (;;)  {
   statement_list;
   if (!condition) break;
   }
```

Thanks