DECISION MAKING & BRANCHING

Introduction

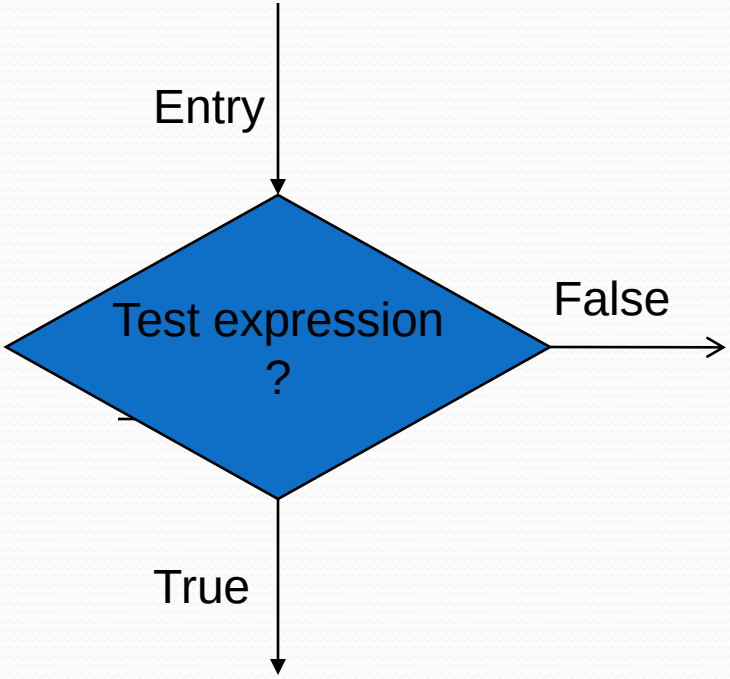
- Instructions of a program are executed either
  - Sequential manner
  - Branching
- C language supports the following decision making statements.
  - if statement
  - switch statement
  - conditional operator
  - goto statement

if statement

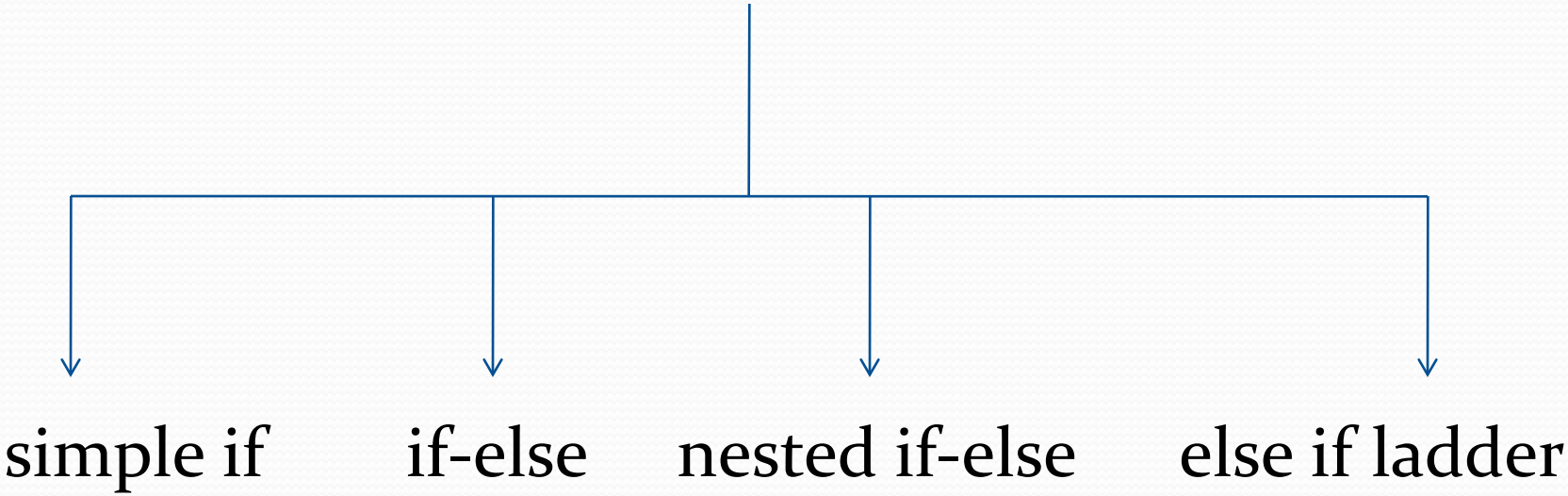
- It is used to control flow of execution of statement.
- It is two-way decision statement and is used in conjuction with an expression/condition

Ex: if (age is more than 55)

Person is retired



Different forms of if statement



Simple if statement

Syntax: if (Condition)

{

statement block;

}

statement x ;

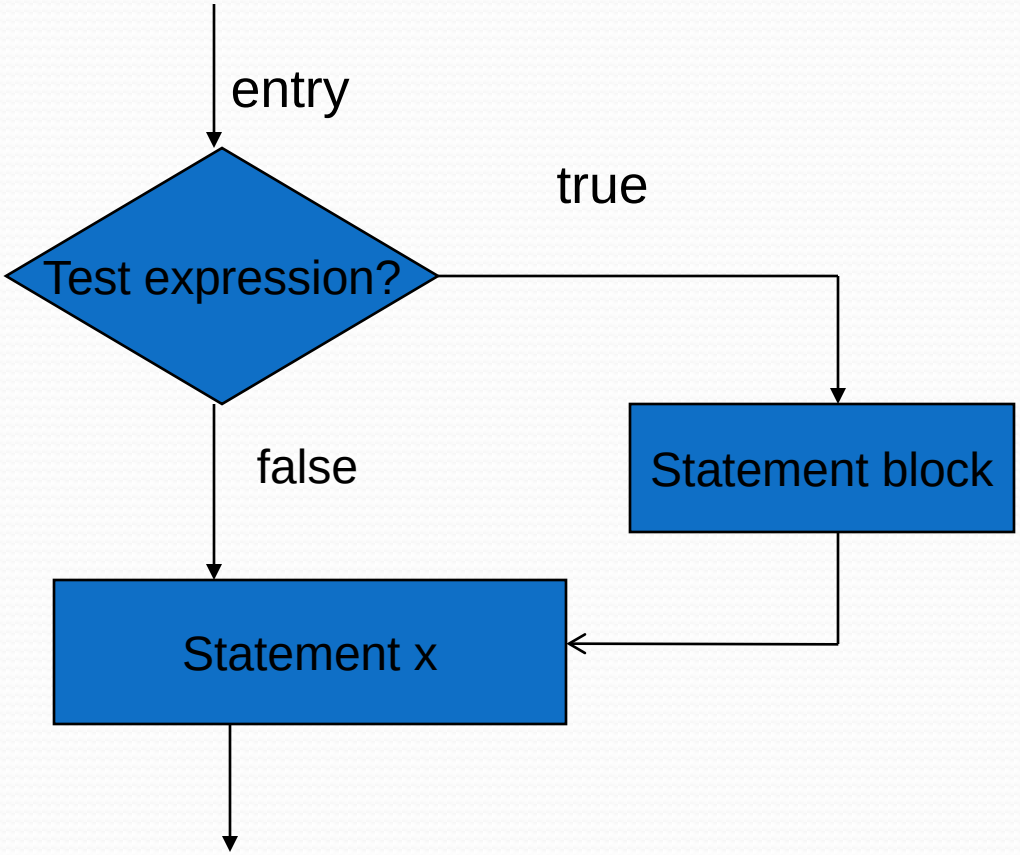
Ex: if (category == sports)

{

marks=marks+bonus;

}

printf (“%f”,marks);



Program of simple *if* statement

```
  main()
{
  int a,b,c,d;
  float ratio;
  printf("\n enter four integer values");
  scanf("%d %d %d %d",&a,&b,&c,&d);
    if(c-d !=0)
     {
      ratio= (float) (a+b)/(float)(c-d);
     }
printf("Ratio= %f", ratio);
 }
```

Output: enter four integer values

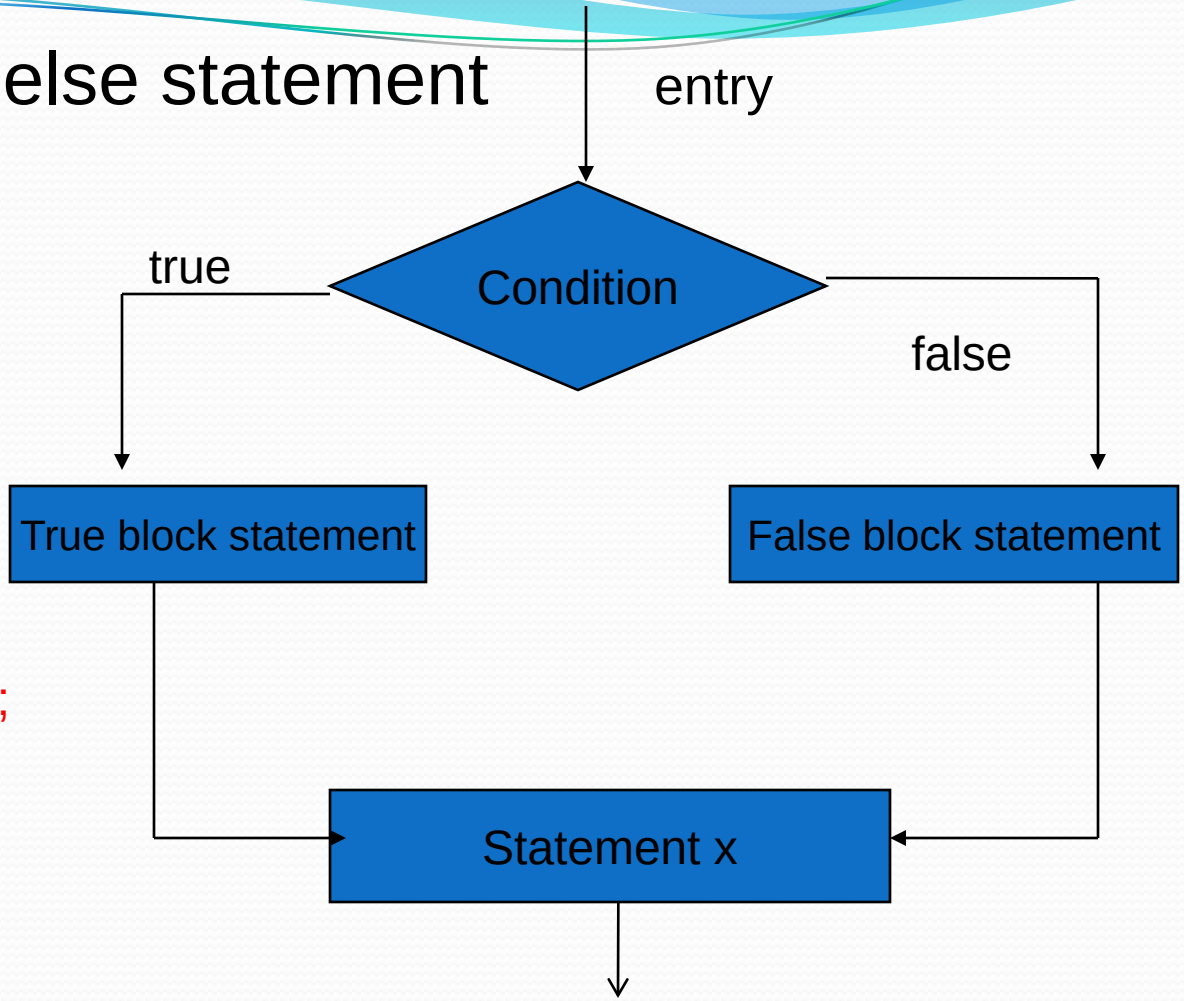
12 23 34 45

Ratio= -3.181818

The if…else statement

Syntax: if(Condition)

```
 {
    true block statement;
 }
else
 {
    false block statement;
 }
statement x;
```



Ex-:

```
if (code== 1)
    boy=boy+ 1;
if ( code== 2)
    girl=girl+1;
```

```
if (code==1)
    boy=boy+1;
else
    girl=girl+1;
```

Program for if…else statement

```
main()
{
  int a;
  printf("Enter an integer\n");
  scanf("%d",&a);
   if(a%2==0)
          printf(" %d is even number",a);
   else
          printf("%d is an odd number",a);
}
```

output- Enter an integer

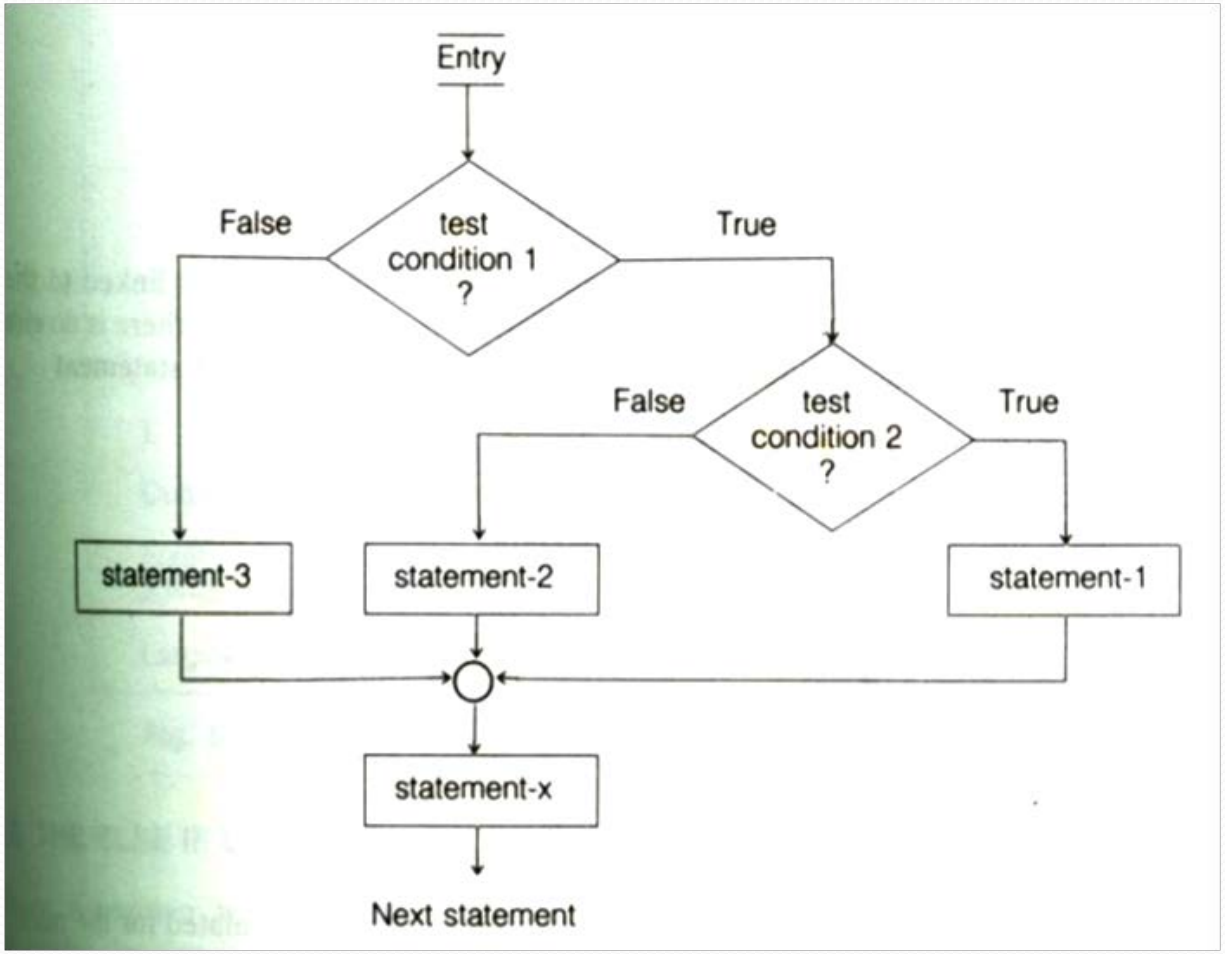
46

46 is an even number

Nested *if…else* statement

Syntax-

```
if( test condition 1)
    {
    if (test condition 2)
      {
            statement 1;
      }
    else
      {
            statement 2;
      }
    }
else
   {
    statement 3;
   }
statement x  ;
```



Program for nested if…else statement

```
main( )
{
      float A, B, C;

      printf("Enter three values\n");
      scanf("%f %f %f", &A, &B, &C);

      printf("\nLargest value is    ");

      if (A>B)
        {
         if (A>C)
            printf("%f\n", A);
         else
            printf("%f\n", C);
        }
       else
        {
         if (C>B)
            printf("%f\n", C);
         else
            printf("%f\n", B);
        }
}
```

Output

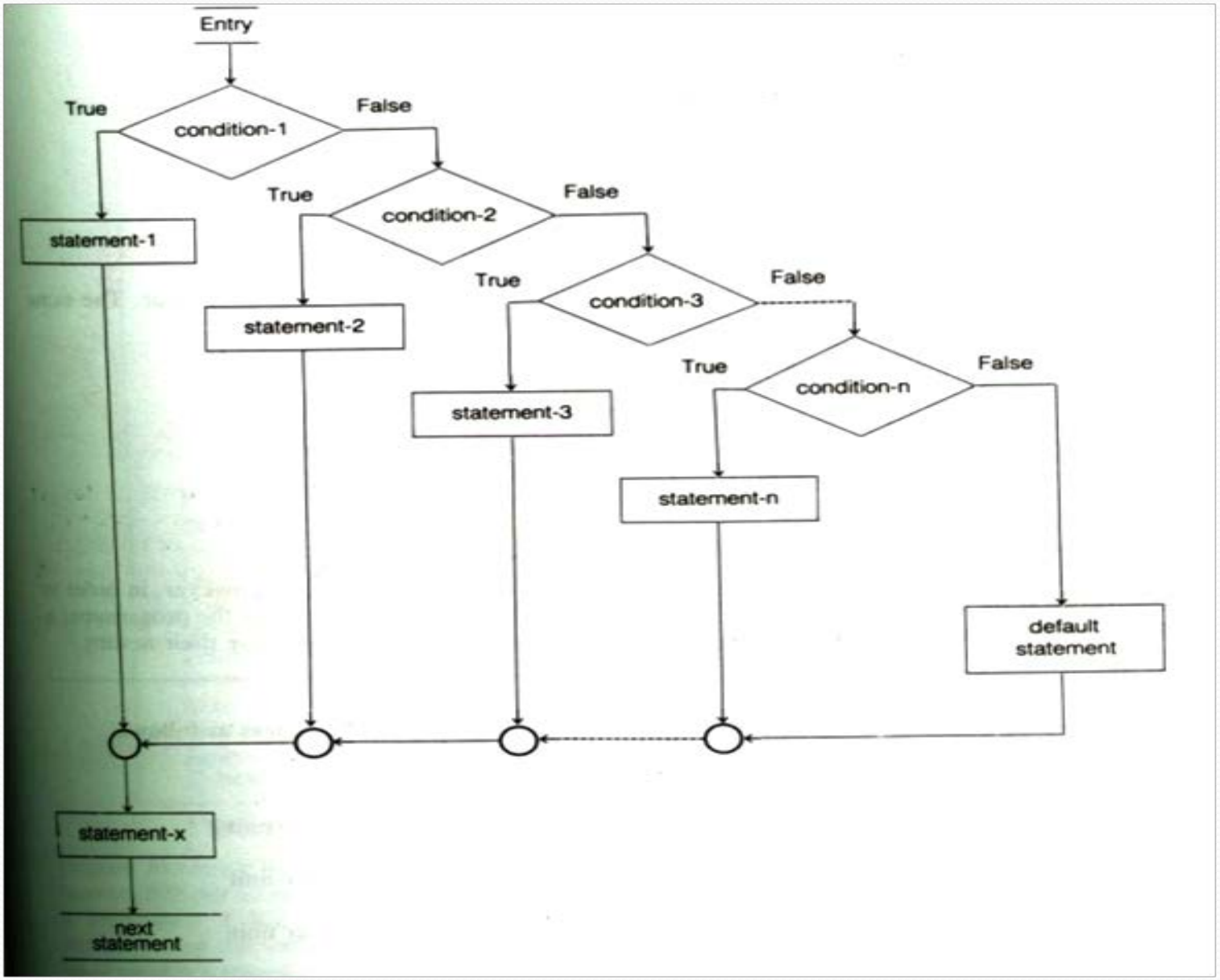
```
Enter three values
23445   67379   88843

Largest value is    88843.000000
```

The else if ladder

syntax-:

```
if  ( condition 1)
  statement 1 ;
     else if ( condition 2)
        statement 2 ;
           else if ( condition 3)
              statement 3 ;
                 else if( condition n)
                    statement n ;
                       else
                          default statement ;
        statement x;
```

Flowchart of else….if ladder

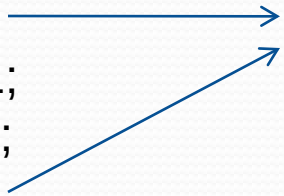


The ***switch*** statement

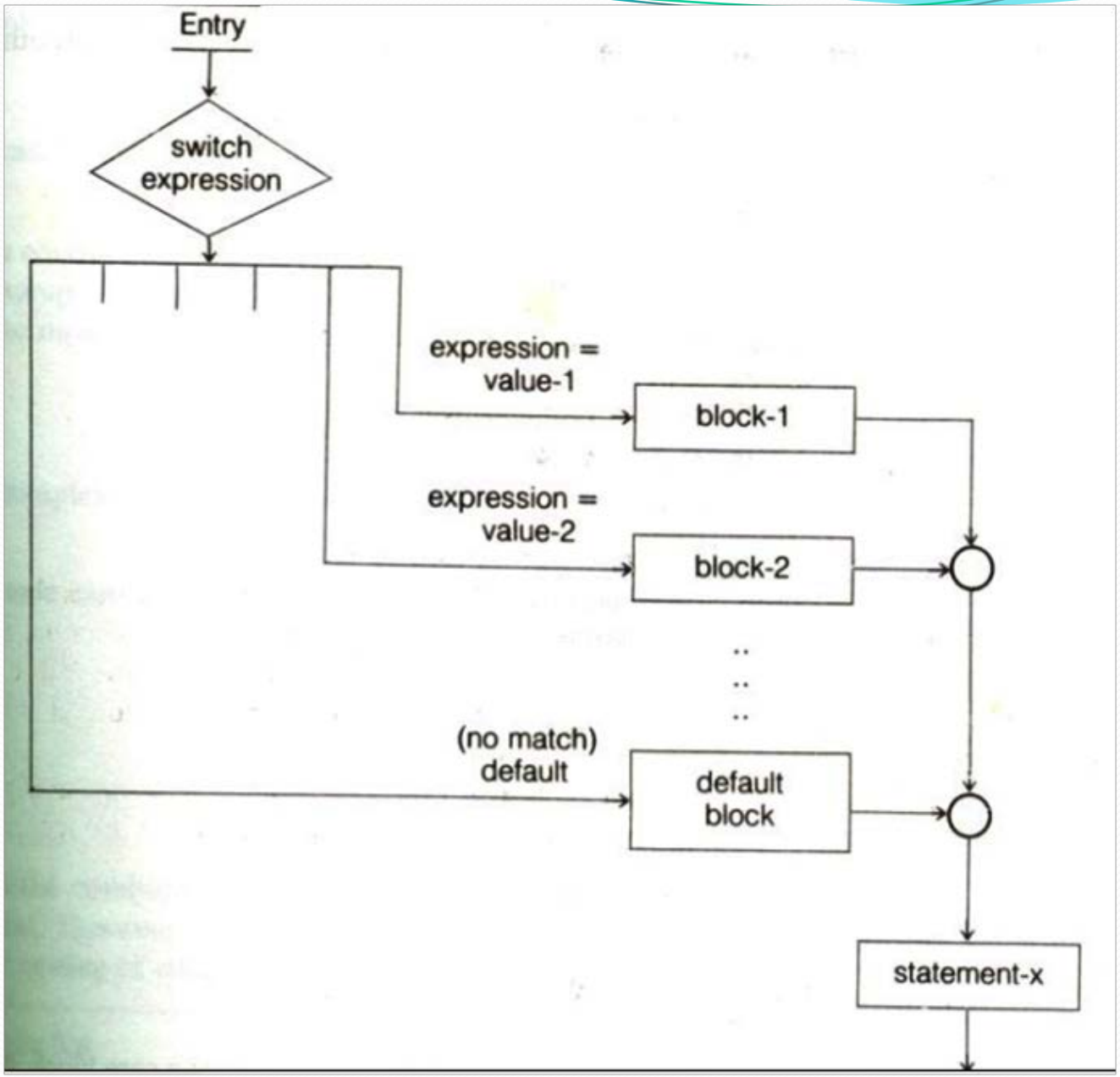
- The complexity of a program increases by increasing no. of if statement.
- To overcome this, C has a built in multi-way decision statement known as switch.

Syntax

```
switch (expression)
    {
case value-1:                    case labels
          block1;
           break;
case value-2:
          block2;
           break;
       .
       .
       .
        default:
             default block;
             break;
    }
 statement x;
```



Flowchart for switch statement



Program for switch statement

```
main()
{
 int grade,mark,index;
 printf("Enter ur mark \n");
 scanf("%d",&mark);
 index=mark/10;
 switch(index)
 {
  case 10:
  case 9:
  case 8:
  case 7:
  case 6:
        grade=1;
        break;
  case 5:
        grade=2;
        break;

   case 4:
        grade=3;
        break;
    default:
         grade=0;
         break;
     }
    printf("The grade is %d",grade);
}
```

The goto statement

- The **goto** statement is used for branching unconditionally.
- The **goto** statement breaks normal sequential execution of the program.
- The **goto** requires label to where it will transfer the control.
- Label is a valid variable name followed by a colon(:).

```
goto label:
  ----------
  ----------
  ----------
  label:
statement;
```

FORWARD JUMP

```
  label:
  statement;
----------
----------
----------
  goto label;
```

BACKWARD JUMP

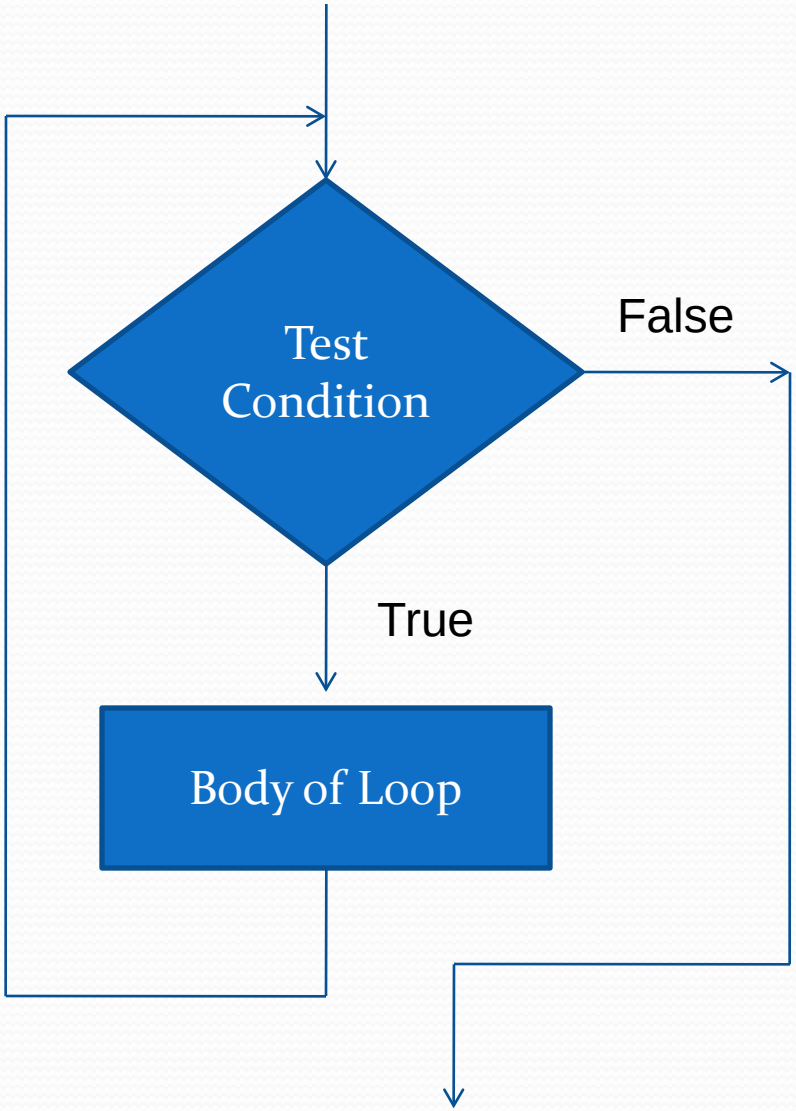
Program to calculate sum of squares of all integers

```
main()
{
 printf(“one\n”);
 printf(“two\n”);
 goto abc;
 printf(“three\n”)
  printf(“four\n”);
  abc:
 printf(“five\n”);
 }
```

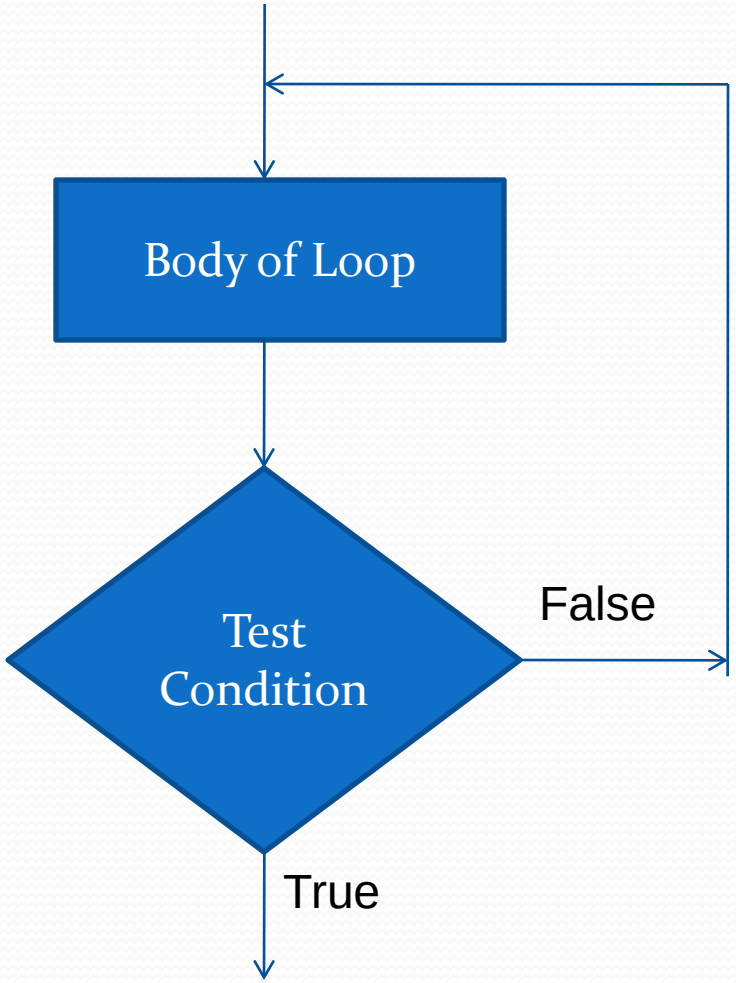
Looping

Loops

- A loop allows a program to repeat a group of statements, either any number of times or until some loop condition occurs
- Convenient if the exact number of repetitions are known
- Loop Consists of
  - Body of the loop
  - Control Statement

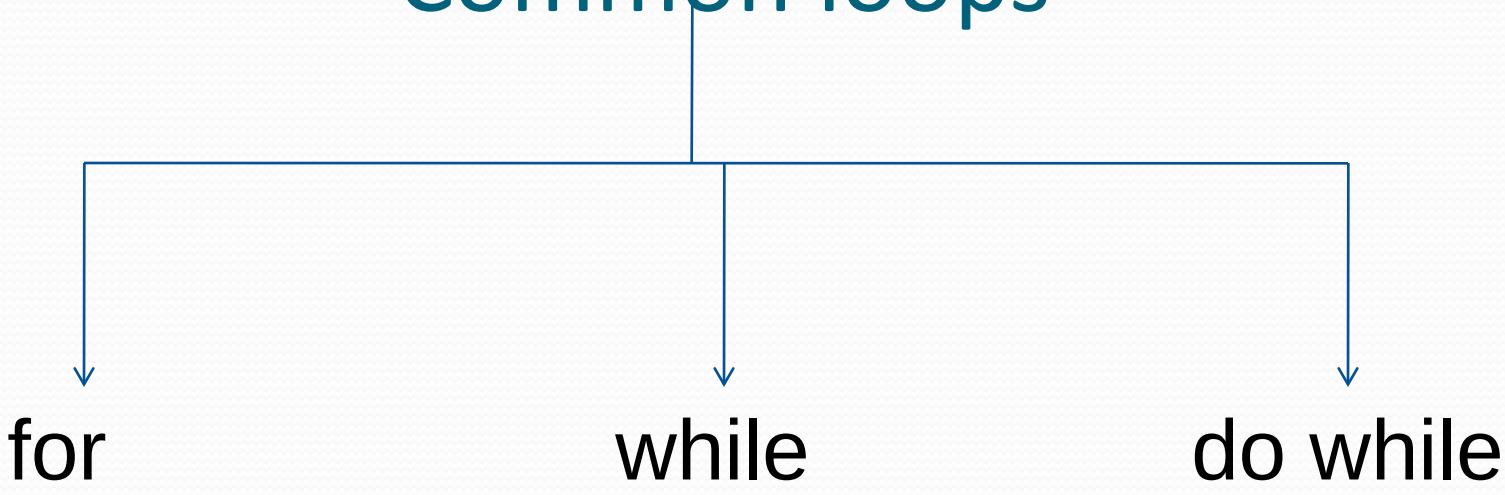


Entry Controlled/Pre-Test Loop



Exit Controlled/Post-Test Loop

Common loops

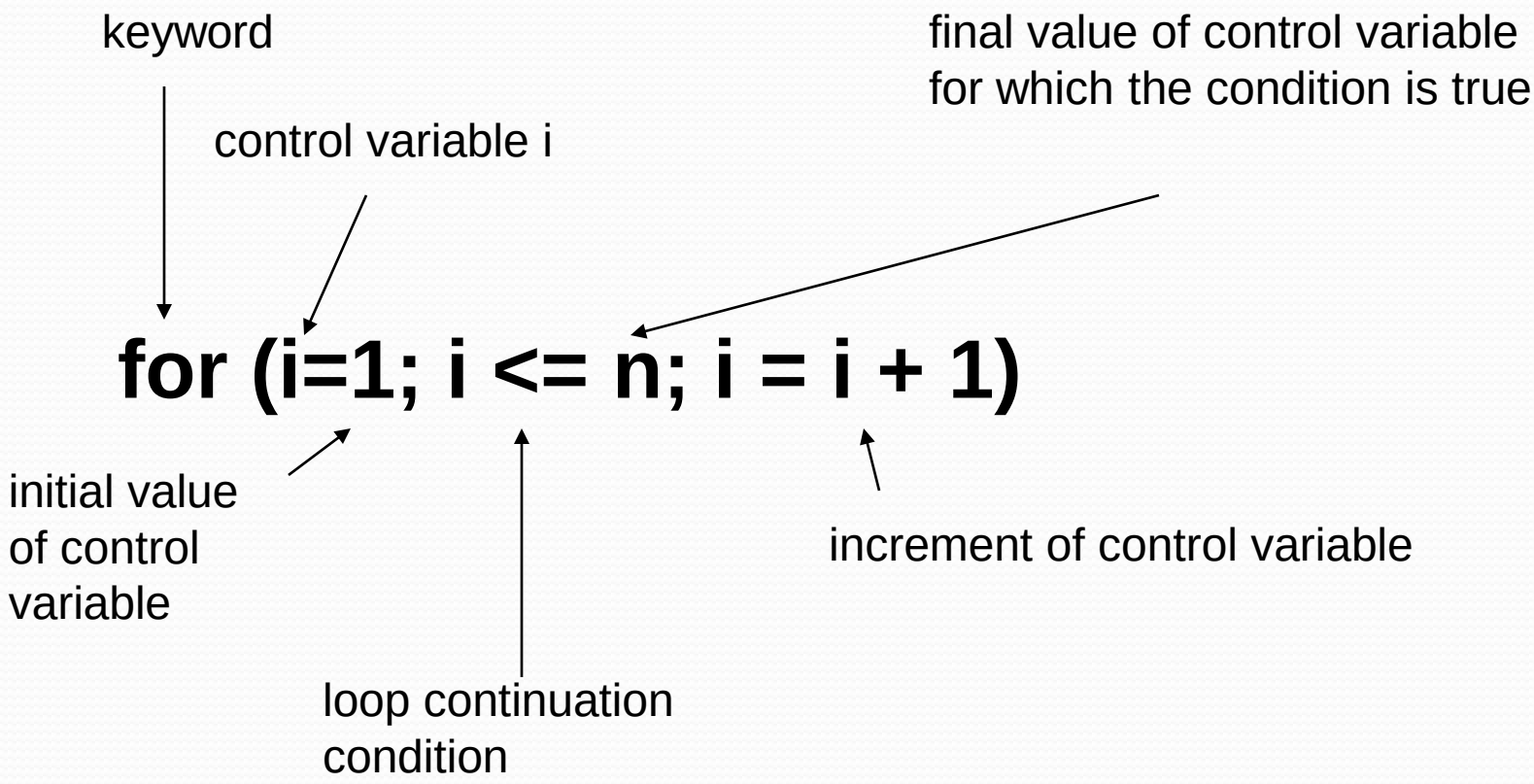


for

Syntax:

```
for (expr1; expr2; expr3)
{
        statement block;
}
```

- expr1 does initializtion of the control variable(s),
- expr2 represents a condition that ensures loop continuation,
- expr3 modifies the value of the control variable(s) initially assigned by expr1


keyword
control variable i
final value of control variable
for which the condition is true
for (i=1; i <= n; i = i + 1)
initial value
of control
variable
loop continuation
condition
increment of control variable

Examples

- Vary the control variable from 1 to 100 in increments of 1

  for (i = 1; i <= 100; i++)

- Vary the control variable from 100 to 1 in increments of -1

  for (i = 100; i >= 1; i=i-1)

- Vary the control variable from 5 to 55 in increments of 5

  for (i = 5; i <= 55; i+=5)

Examples 2

```
#include <stdio.h>
main()
{
    /* a program to produce a Celsius to Fahrenheit conversion chart for the
    numbers 1 to 100 */
    int celsius;
    for (celsius = 0; celsius <= 100; celsius++)
        printf("Celsius: %d Fahrenheit: %d\n", celsius, (celsius * 9) / 5 + 32);
    }
```

Nested **for** loops

- Nested means there is a loop within a loop
- Executed from the inside out
  - Each loop is like a layer and has its own counter variable, its own loop expression and its own loop body
  - In a nested loop, for each value of the outermost counter variable, the complete inner loop will be executed once

General form

```
for (loop1_exprs) {
      statment p;
      statement q;

     for (loop2_exprs) {
     statements for loop2
     }
      statment y;
      statement z;
}
```

while

```
expression 1;        ⟶ initialization of condition variables
 while (expression 2)  ⟶ condition checking
   {
   statement block;
   expression 3;⟶ updates of condition variables
   }
   Statement X;
```

- The statement is executed repeatedly as long as the expression2 is true (non zero)

- When the expression becomes false, the execution resumes from statement X.

- If the test expression in a while loop is false initially, the while loop will never be executed

```
int i = 1, sum = 0;
while (i <= 10)
{
        sum = sum + i;
         i= i + 1;
}
printf("Sum = %d\n", sum);
```

for and **while**

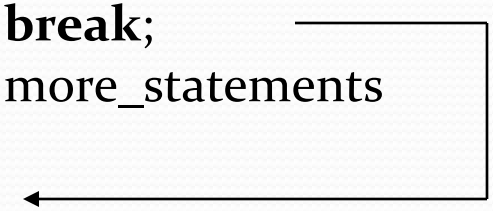
```
for(expr1; expr2; expr3)
        statement;
```

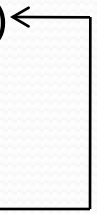
```
 expr1;
while(expr2)
{
        statement;
         expr3;
}
```

break and **continue**

- These interrupt normal flow of control
- **break** causes an exit from the innermost enclosing loop
- **continue** causes the current iteration of a loop to stop and the next iteration to begin immediately

```
while (expression)
{
    statements;
    if(condition)
    break;
    more_statements
}

while (expression)
{
    statements
    continue;
    more_statements
}
```





do-while

- When a loop is constructed using while, the test for continuation is carried out at the beginning of each pass

- With do-while the test for continuation takes place at the end of each pass

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

Example

```
int i = 1, sum = 0;
   do
   {
         sum = sum + i;
           i= i + 1;
   }while(i<=10);
   printf("Sum = %d\n", sum);
```

while vs. do-while

- **while** -- the expression is tested first, if the result is false, the loop body is never executed

- **do-while** -- the loop body is always executed once. After that, the expression is tested, if the result is false, the loop body is not executed again