Structured Programming Language

Lecture 4

Introduction

- C Language supports the following decision-making statements:
  - **if** statement
  - **switch** statement
  - Conditional operator statement
  - **goto** statement
  - **while** statement
  - **for** statement
  - **do…while** statement

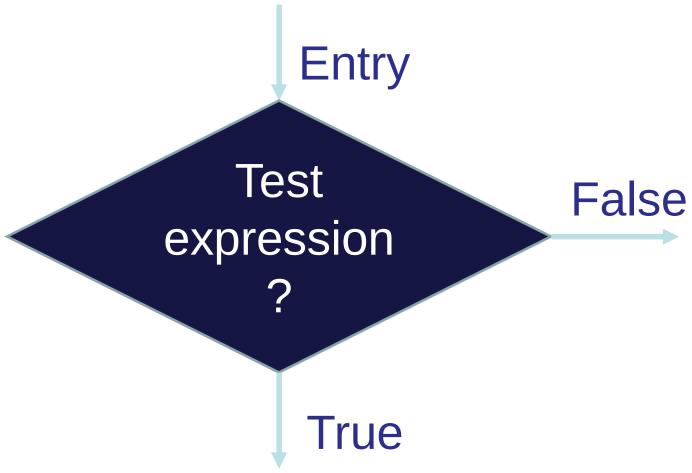
Decision making with if statement

- **if** is used to control the flow of the execution of statements.

If (test expression)

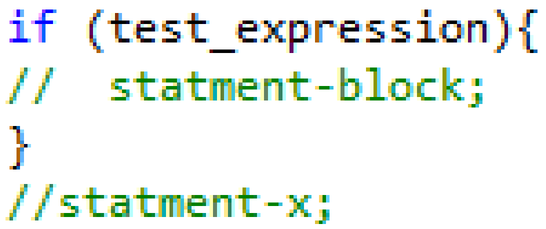
- It allows the computer to evaluate the expression first and them depending on whether the value of the expression (relation of condition) is true (non-zero) of false (zero) it transfers the control to a particular statement.

The **if** statement is called *branching statement of selection statement* because it provide a junction where the program has to select which of two paths to follow.



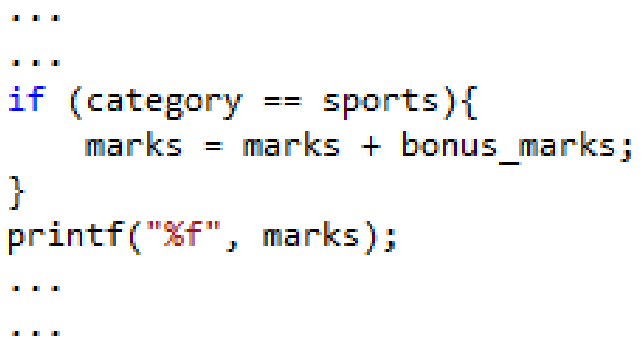
1. if (bank balance is zero)
   borrow money
2. if (room is dark)
   turn on lights
3. if (code is 1)
   person is male
4. if (age is more than 55)
   person is retired

Simple if statement



```
if (test_expression){
//  statment-block;
}
//statment-x;
```

- The statement-block may be a single a statement or a group of statements.
- If the **test_expression** is true, the statement-block will be executed; otherwise the **statement-block** will be skipped and the execution will jump to the **statement-x**.

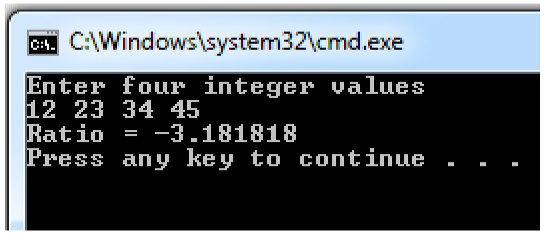


```
...
...
if (category == sports){
    marks = marks + bonus_marks;
}
printf("%f", marks);
...
...
```

The program tests the type of category of the student. If the student belongs to the SPORTS category, then additional bonus_marks are added to his marks before that is printed.

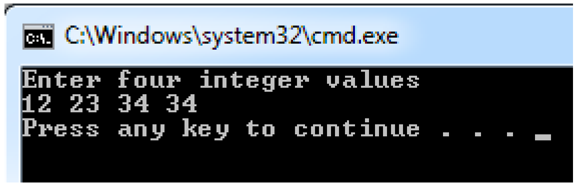
Illustration of simple if statement

```c
void main(){

    int a, b, c ,d;
    float ratio;

    printf("Enter four integer values\n");
    scanf("%d %d %d %d", &a, &b, &c, &d);

    if(c - d != 0) //execute statement block
    {
        ratio = (float) (a + b) / (float (c - d));
        printf("Ratio = %f\n", ratio);
    }

}
```



C:\Windows\system32\cmd.exe

```
Enter four integer values
12 23 34 45
Ratio = -3.181818
Press any key to continue . . .
```



C:\Windows\system32\cmd.exe

```
Enter four integer values
12 23 34 34
Press any key to continue . . .
```

The result of the first run is printed as
Ratio = -3.181818

The <u>second run</u> has neither produced any results nor any message.
During the second run, the value (c - d) is equal to zero and therefore, the statements contained in the statement block are skipped.

<u>To avoid truncation due to integer division.</u>

The shoes1.c program

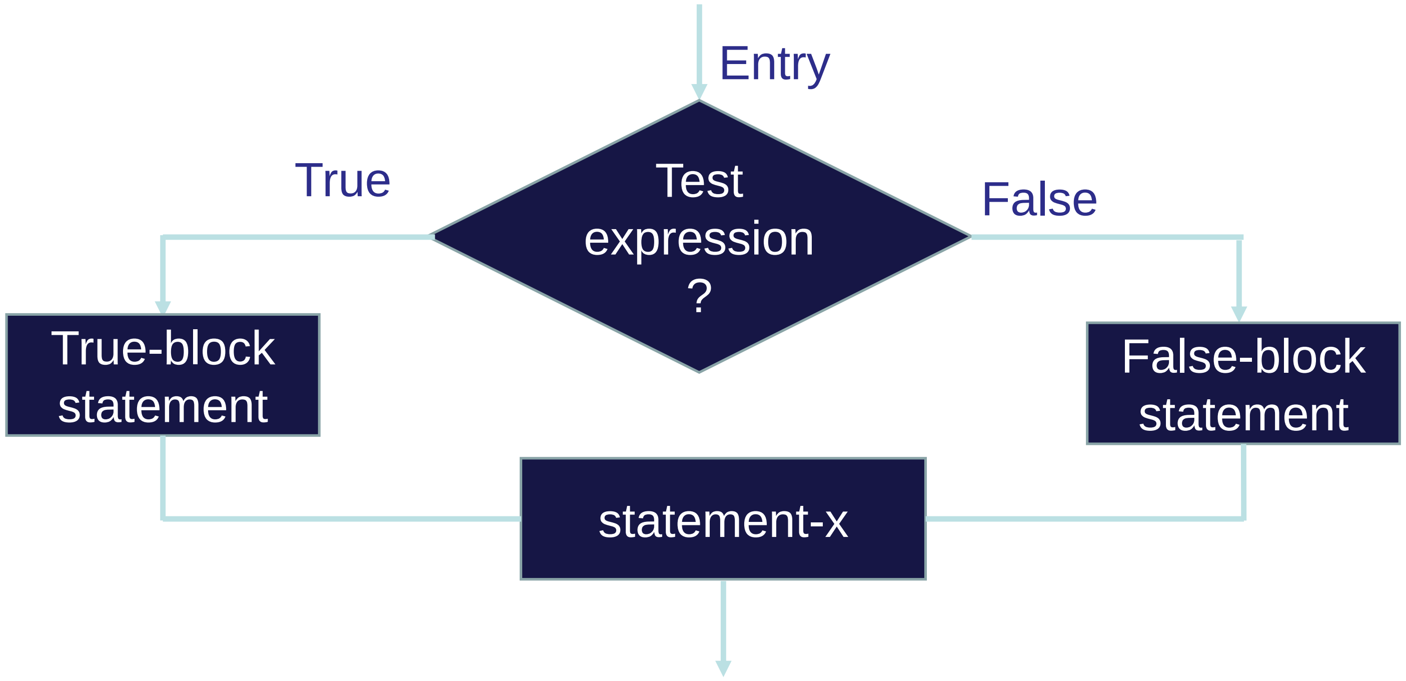
```c
/* shoes1.c -- converts a shoe size to inches */
#include <stdio.h>
#define ADJUST 7.64
#define SCALE 0.325
int main(void)
{
    double shoe, foot;

    shoe = 9.0;
    foot = SCALE * shoe + ADJUST;
    printf("Shoe size (men's)    foot length\n");
    printf("%10.1f %15.2f inches\n", shoe, foot);

    return 0;
}
```

- The program takes your shoes size and tells you how long your foot is in inches.

The **if** … **else** statement

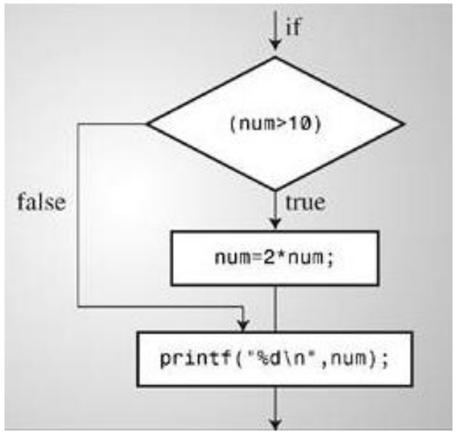
```
if (test_expression){
    //True-block statment(s)
}else{
    //False-block statment(s)
}
//statment-x
```

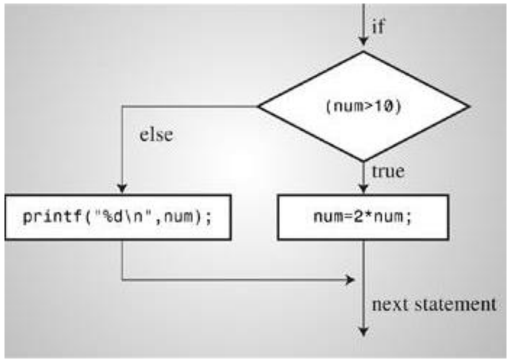
- If the test expression is true, then the *true-block statement(s)*, immediately following the if statements are executed; otherwise, *the false-block statement(s)* are executed.
- In either case, either *true-block* or *false-block* will be executed, not both.



Adding else to the if Statement

C enables to choose between two statements by using the **if else** form.





```
if (all_days != 0)
    printf("%d days total: %.1f%% were below freezing.₩n",
           all_days, 100.0 * (float) cold_days / all_days);

if (all_days == 0)
    printf("No data entered!₩n");
```

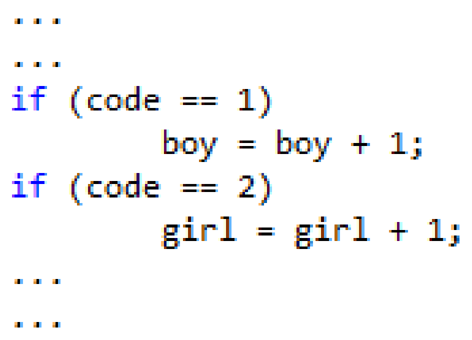
if (*expression*)
statement1
else
statement2

```
if (all_days!= 0)
    printf("%d days total: %.1f%% were below freezing.₩n",
           all_days, 100.0 * (float) cold_days / all_days);
else
       printf("No data entered!₩n");
```

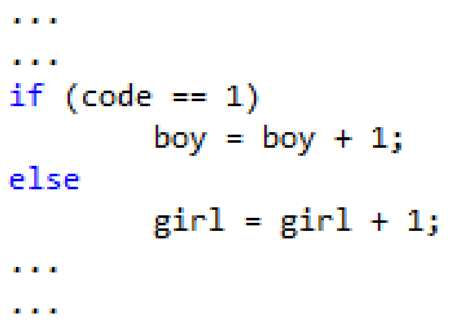
No need for second **If** →

Examples

- The test determines whether or not the student is a boy or girl and increment corresponding variable.

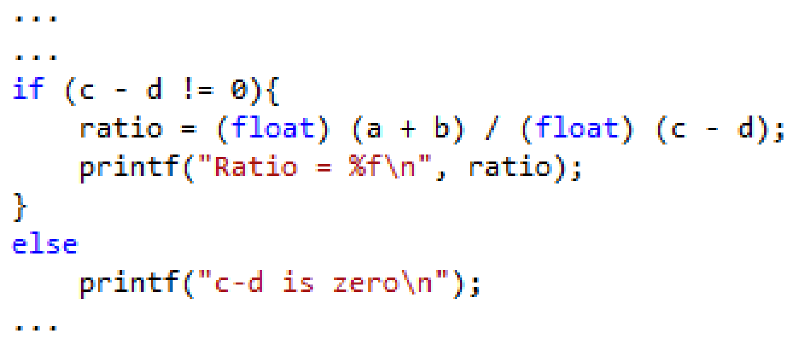


```
...
...
if (code == 1)
        boy = boy + 1;
if (code == 2)
        girl = girl + 1;
...
...
```



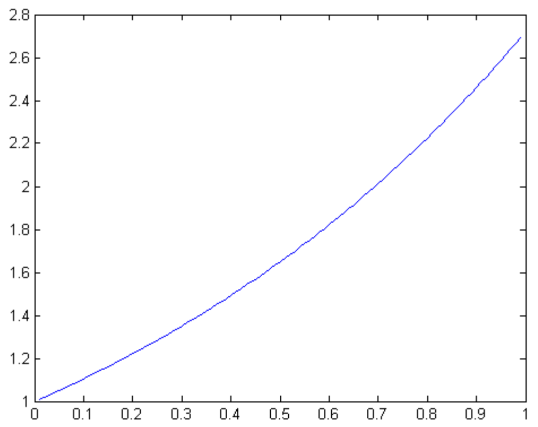
```
...
...
if (code == 1)
        boy = boy + 1;
else
        girl = girl + 1;
...
...
```

- When the value (c – d) is zero, the ratio is not calculated and the program stops without any message. In such cases we may not know whether the program stopped due to a zero value of some other error.



```
...
...
if (c - d != 0){
    ratio = (float) (a + b) / (float) (c - d);
    printf("Ratio = %f\n", ratio);
}
else
    printf("c-d is zero\n");
...
```

Example



- A program evaluates the power series
- $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}, 0 < x < 1.$

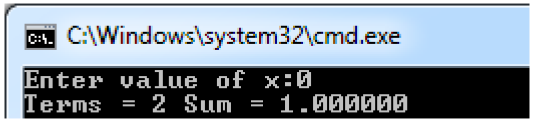
```c
#include <stdio.h>
#define ACCURACY 0.0001

void main(){

    int n, count;
    float x, term, sum;
    printf("Enter value of x:");
    scanf("%f", &x);
    n = term = sum = count = 1;

    while (n <= 100){
        term *= x/n;// term = term * x/n;
        sum += term; // sum = sum + term;
        count++;
        if (term < ACCURACY)
            n = 999;
        else
            n++;
    }
    printf ("Terms = %d Sum = %f\n", count, sum);

}
```

- The power series contains the recurrence relationship of the type $T_n = T_{n-1}\left(\frac{x}{n}\right)$ for $n > 1$.

$T_1 = x$ for $n = 1$, $T_0 = 1$

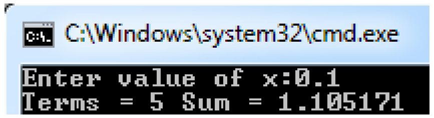
If T_{n-1} is known, then T_n can be easily found by multiplying the previous term by x/n.

Then $e^x = T_0 + T_1 + T_2 + \cdots + T_n = sum$



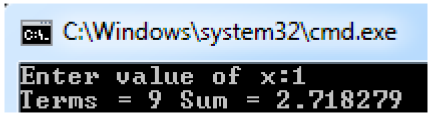
C:\Windows\system32\cmd.exe

```
Enter value of x:0
Terms = 2 Sum = 1.000000
```



C:\Windows\system32\cmd.exe

```
Enter value of x:0.1
Terms = 5 Sum = 1.105171
```



C:\Windows\system32\cmd.exe

```
Enter value of x:1
Terms = 9 Sum = 2.718279
```

Use braces to create a single block

If you want more than one statement between the if and the else, you must use braces to create a single block.

Wrong

```
if (x > 0)
    printf("Incrementing x:₩n");
    x++;
else            // will generate an error
    printf("x <= 0 ₩n");
```

Right

```
if (x > 0)
{
    printf("Incrementing x:₩n");
    x++;
}
else
    printf("x <= 0 ₩n");
```

Nesting of if…else statements

- When a series of decisions are involved, we may have to use more than one **if**…**else** statement in nested form.

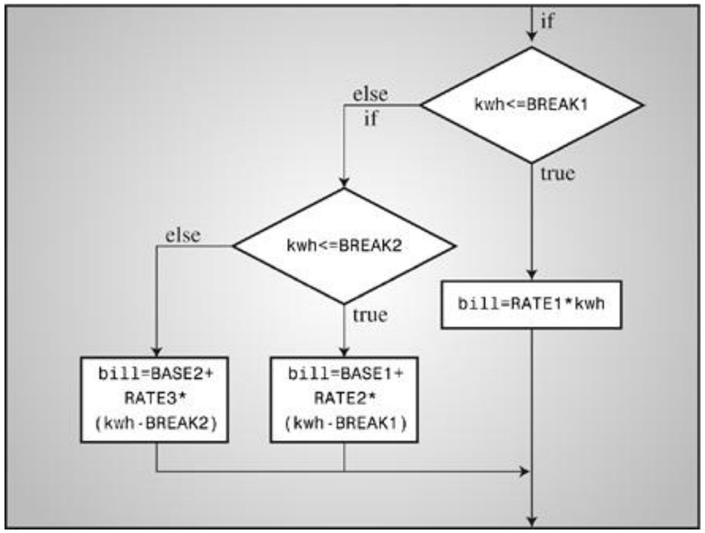
Example: Here are the rates one company charges for electricity, based on kilowatt-hours (kWh):

First 360 kWh:	$0.12589 per kWh
Next 320 kWh:	$0.17901 per kWh
Over 680 kWh:	$0.20971 per kWh

Let’s prepare a program to calculate our energy costs using multiple choice statement.

The electric.c Program

```c
#include <stdio.h>

#define RATE1   0.12589       /* rate for first 360 kwh      */
#define RATE2   0.17901       /* rate for next 320 kwh       */
#define RATE3   0.20971       /* rate for over 680 kwh       */
#define BREAK1  360.0         /* first breakpoint for rates  */
#define BREAK2  680.0         /* second breakpoint for rates */
#define BASE1   (RATE1 * BREAK1)
                              /* cost for 360 kwh            */
#define BASE2  (BASE1 + (RATE2 * (BREAK2 - BREAK1)))
                              /* cost for 680 kwh            */
int main(void)
{
    double kwh;               /* kilowatt-hours used         */
    double bill;              /* charges                     */

    printf("Please enter the kwh used.₩n");
    scanf("%lf", &kwh);       /* %lf for type double         */

    if (kwh <= BREAK1)
        bill = RATE1 * kwh;
    else if (kwh <= BREAK2)   /* kwh between 360 and 680     */
        bill = BASE1 + (RATE2 * (kwh - BREAK1));
    else                      /* kwh above 680               */
        bill = BASE2 + (RATE3 * (kwh - BREAK2));

    printf("The charge for %.1f kwh is $%1.2f.₩n", kwh, bill);

    return 0;
}
```



if
kwh<=BREAK1
else if
true
kwh<=BREAK2
else
true
bill=RATE1*kwh
bill=BASE2+ RATE3* (kwh-BREAK2)
bill=BASE1+ RATE2* (kwh-BREAK1)

```
Please enter the kwh used.
580
The charge for 580.0 kwh is
$84.70.
```

The **else if** is a variation on what you already knew

```
if (kwh <=BREAK1)

     bill = RATE1 * kwh;

else

    if (kwh <=BREAK2)

        bill = BASE1 + RATE2 * (kwh - BREAK1);

    else

        bill = BASE2 + RATE3 * (kwh - BREAK2);
```

```
if (kwh <= BREAK1)

    bill = RATE1 * kwh;

else if (kwh <= BREAK2)   /* kwh between 360 and 680      */

    bill = BASE1 + (RATE2 * (kwh - BREAK1));

else                      /* kwh above 680                */

    bill = BASE2 + (RATE3 * (kwh - BREAK2));
```

You can string together as many else if statements as you need

```
if (score < 1000)
      bonus = 0;
else if (score < 1500)
      bonus = 1;
else if (score < 2000)
      bonus = 2;
else if (score < 2500)
      bonus = 4;
else
      bonus = 6;
```

Pairing else with if

When you have a lot of ifs and elses, how does the computer decide which if goes with which else?

For example, consider the following program fragment:

```
if (number > 6)
    if (number < 12)
       printf("You're close!₩n");
else
    printf("Sorry, you lose a turn!₩n");
```

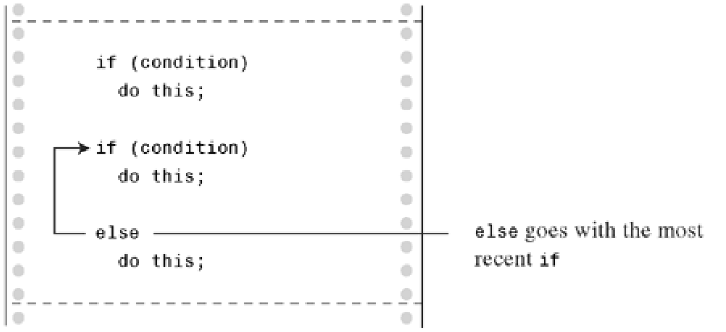
When is Sorry, you lose a turn! printed?

a. When number is less than or equal to 6, or

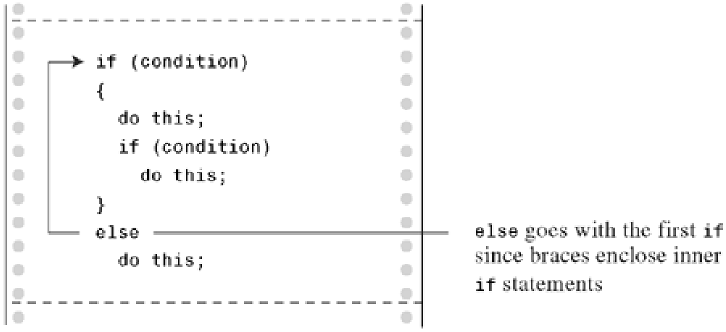
b. When number is greater than 12?

In other words, does the else go with the first if or the second?

The answer is, the else goes with the second if.



```
if (condition)
  do this;

if (condition)
  do this;

else
  do this;
```



```
if (condition)
{
  do this;
  if (condition)
    do this;
}
else
  do this;
```

else goes with the first if since braces enclose inner if statements

```
if (number > 6)
    if (number < 12)
       printf("You're close!₩n");
else
    printf("Sorry, you lose a turn!₩n");
```

≠

```
if (number > 6)
{
      if (number < 12)
         printf("You're close!₩n");
}
else
   printf("Sorry, you lose a turn!₩n");
```

The ? : Operator

- We have covered conditional operator ? : in the previous chapter which can be used to replace if...else statements. It has the following general form −

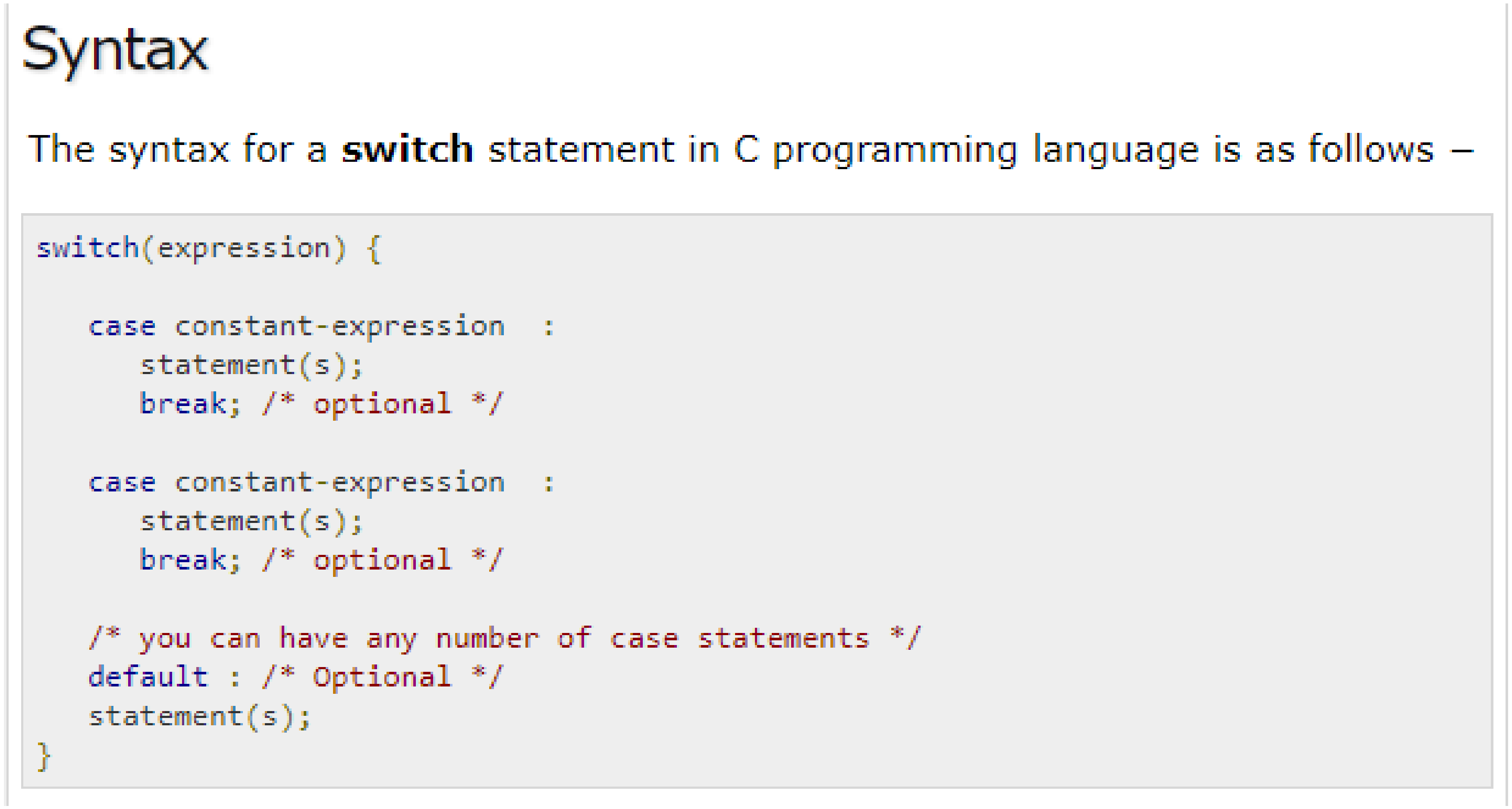
  Exp1 ? Exp2 : Exp3;

- Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.
- The value of a ? expression is determined like this −
  - Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.
  - If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

Switch Statement

- A **switch** statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each **switch case**.

Switch Cont.



Syntax

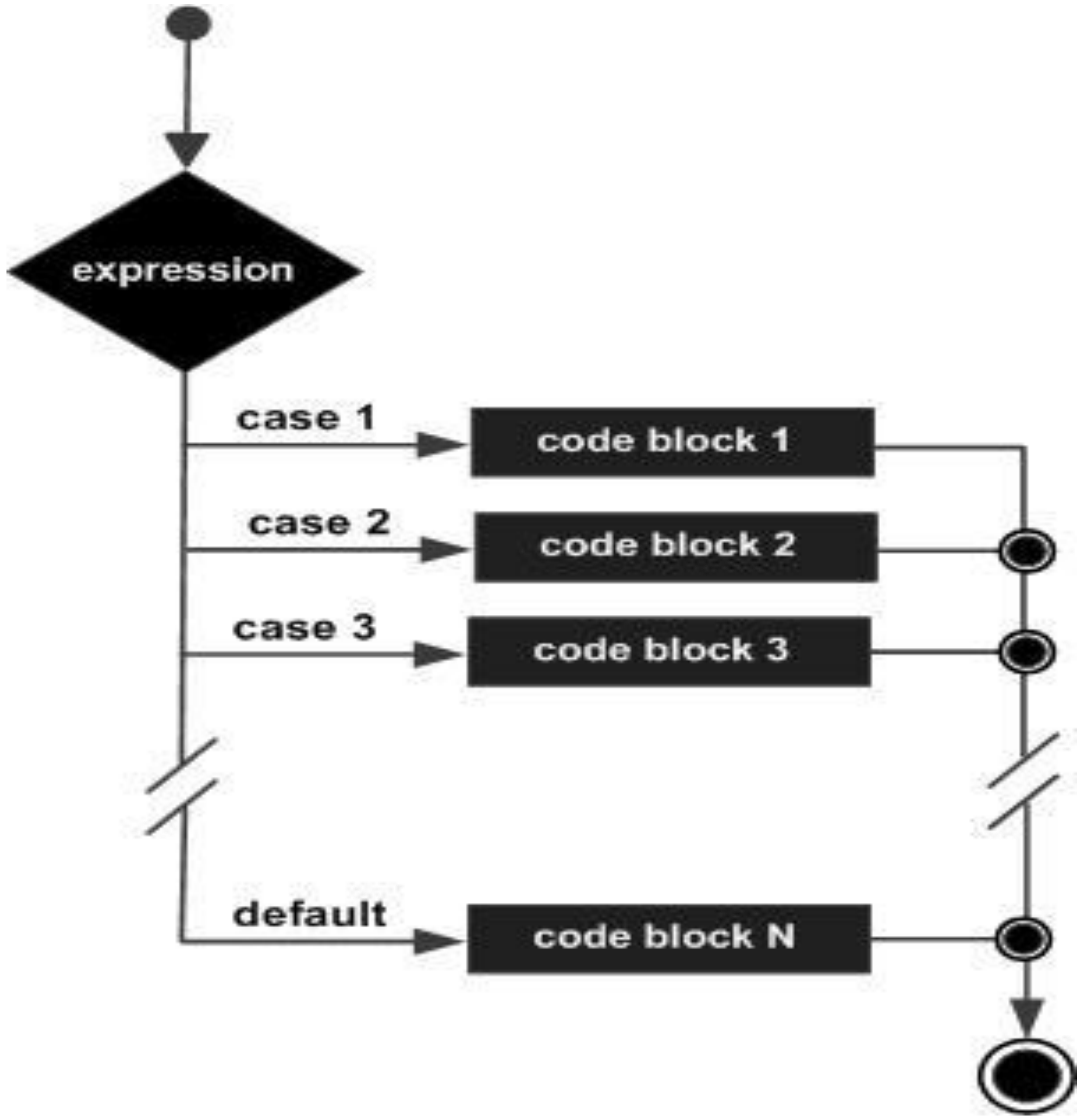
The syntax for a **switch** statement in C programming language is as follows –

```
switch(expression) {

   case constant-expression  :
      statement(s);
      break; /* optional */

   case constant-expression  :
      statement(s);
      break; /* optional */

   /* you can have any number of case statements */
   default : /* Optional */
   statement(s);
}
```

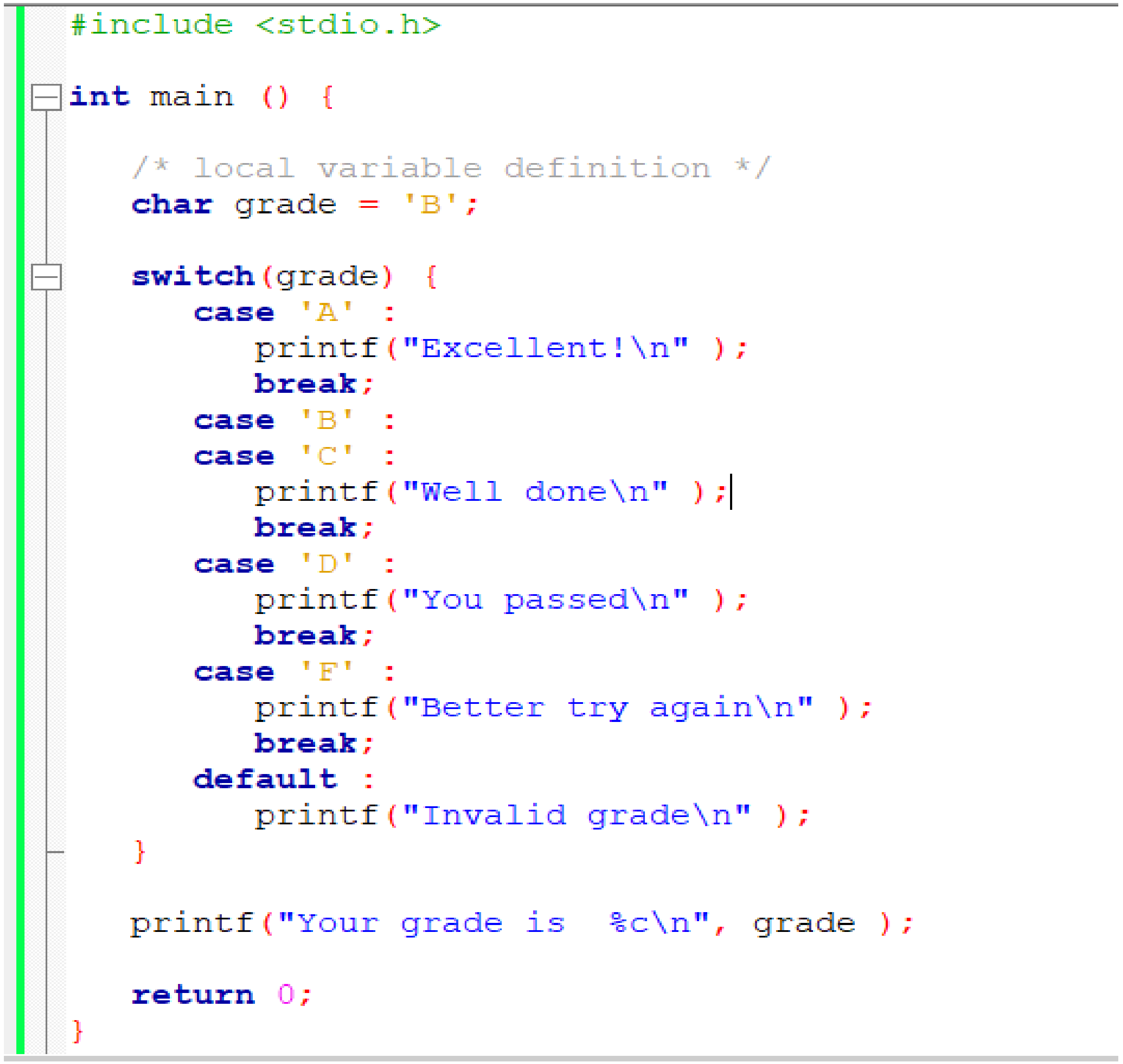
Switch Cont.

- The following rules apply to a **switch** statement −
  - The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
  - You **can have any number of case** statements within a switch. **Each case is followed by the value to be compared to and a colon**.
  - The **constant-expression** for a case must be the **same data type as the variable in the switch**, and it must be a constant or a literal.
  - When the variable being switched on is equal to a case, **the statements following that case will execute until a break** statement is reached.
  - When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
  - Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
  - A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

Switch Flow Diagram



Switch Cont.



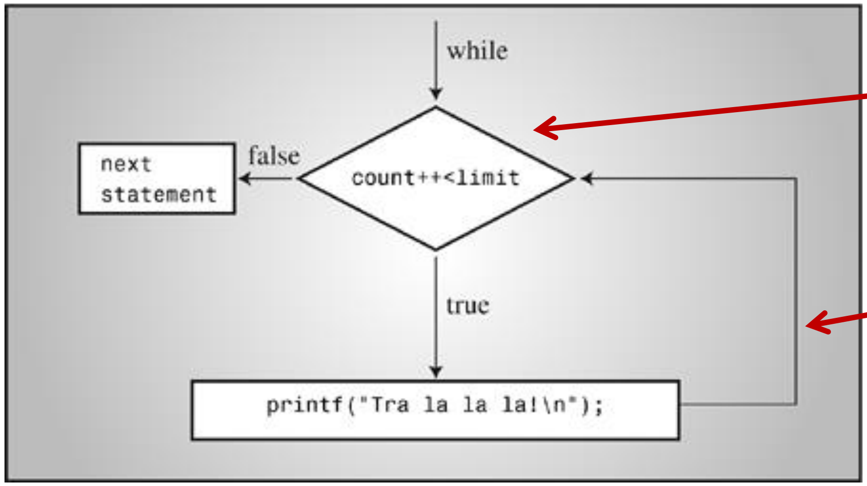
```
#include <stdio.h>

int main () {

   /* local variable definition */
   char grade = 'B';

   switch(grade) {
      case 'A' :
         printf("Excellent!\n" );
         break;
      case 'B' :
      case 'C' :
         printf("Well done\n" );
         break;
      case 'D' :
         printf("You passed\n" );
         break;
      case 'F' :
         printf("Better try again\n" );
         break;
      default :
         printf("Invalid grade\n" );
   }

   printf("Your grade is  %c\n", grade );

   return 0;
}
```

Loop

- You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages −

The while Statement

- The statement part can be a simple statement with a terminating semicolon, or it can be a compound statement enclosed in braces.



This cycle of test and execution is repeated until expression becomes false (zero).

Each cycle is called an *iteration*.

The while loop: shoes2.c

- When the program first reaches the while statement, it checks to see whether the condition within parentheses is true.

  shoe < 18.5

- The < symbol means "is less than." The variable shoe was initialized to 3.0, which certainly is less than 18.5. Therefore, the condition is true and the program proceeds to the next statement.

  shoe = shoe + 1.0;

- The program returns to the while portion to check the condition (because the next line is a closing brace (})).
- This continues until shoe reaches a value of 19.0. Now the condition

  shoe < 18.5

- becomes false because 19.0 is not less than 18.5. When this happens, control passes to the first statement following the while loop.

```
/* shoes2.c -- calculates foot lengths for several sizes */
#include <stdio.h>
#define ADJUST 7.64
#define SCALE 0.325
int main(void)
{
    double shoe, foot;

    printf("Shoe size (men's)    foot length\n");
    shoe = 3.0;
    while (shoe < 18.5)      /* starting the while loop */
    {                        /* start of block          */
        foot = SCALE*shoe + ADJUST;
        printf("%10.1f %15.2f inches\n", shoe, foot);
        shoe = shoe + 1.0;
    }                        /* end of block            */
    printf("If the shoe fits, wear it.\n");

    return 0;
}
```

The shoes2.c Program

```c
/* shoes2.c -- calculates foot lengths for several sizes */
#include <stdio.h>
#define ADJUST 7.64
#define SCALE 0.325
int main(void)
{
    double shoe, foot;

    printf("Shoe size (men's)    foot length\n");
    shoe = 3.0;
    while (shoe < 18.5)      /* starting the while loop */
    {                        /* start of block          */
        foot = SCALE*shoe + ADJUST;
        printf("%10.1f %15.2f inches\n", shoe, foot);
        shoe = shoe + 1.0;
    }                        /* end of block            */
    printf("If the shoe fits, wear it.\n");

    return 0;
}
```

```
Shoe size (men's)    foot length
       3.0              8.62 inches
       4.0              8.94 inches
       ...              ...
      17.0             13.16 inches
      18.0             13.49 inches
If the shoe fits, wear it.
```

Example

The program reads in a list of daily low temperatures and reports the total number of entries and the percentage that were below freezing.

```
#include <stdio.h>
int main(void)
{

    const int FREEZING = 0;
    float temperature;
    int cold_days = 0;
    int all_days = 0;

    printf("Enter the list of daily low temperatures.\n");
    printf("Use Celsius, and enter q to quit.\n");

    while (scanf("%f", &temperature) == 1)
    {
        all_days++;
        if (temperature < FREEZING)
            cold_days++;
    }

    if (all_days != 0)
        printf("%d days total: %.1f%% were below freezing.\n",
               all_days, 100.0 * (float) cold_days / all_days);

    if (all_days == 0)
        printf("No data entered!\n");

    return 0;
}
```

<u>A sample run:</u>

```
Enter the list of daily low temperatures. Use Celsius, and enter q to quit.
12 5 -2.5 0 6 8 -3 -10 5 10 q
10 days total: 30.0% were below freezing.
```

for Loop

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to **execute a specific number of times.**

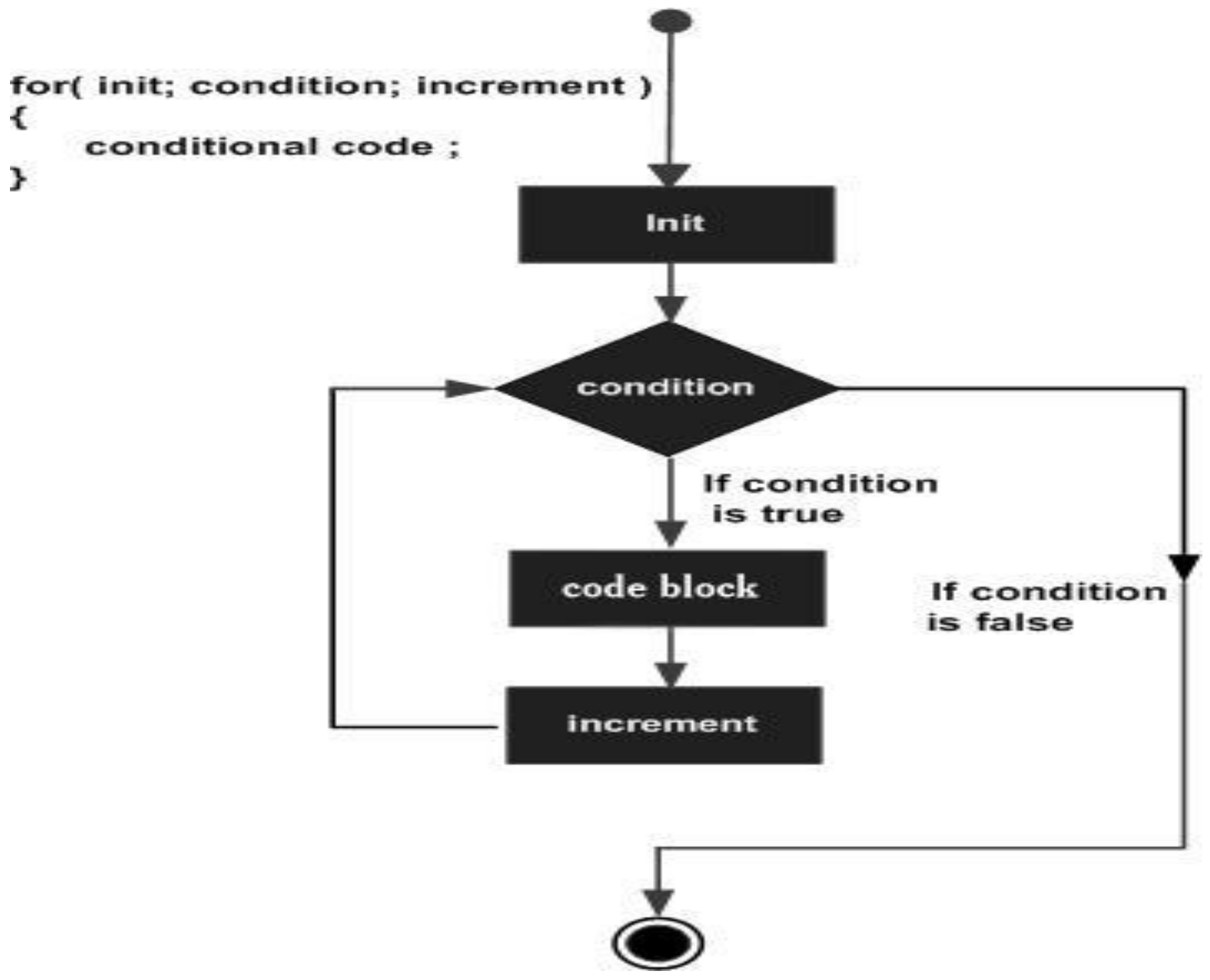
- The syntax of a for loop in C programming language is −

```
for ( init; condition; increment ) {
   statement(s);
}
```

for Loop Cont.

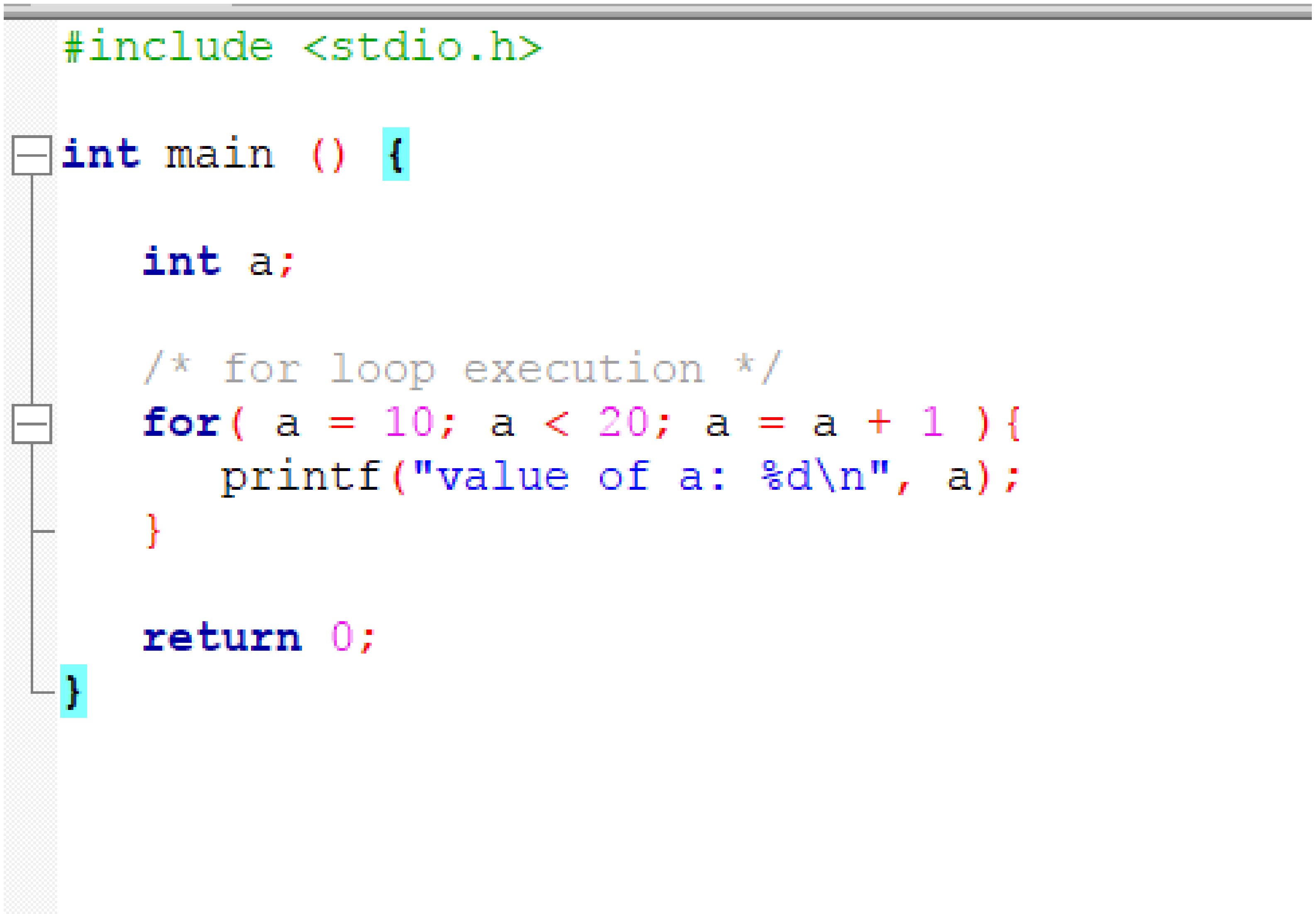
- Here is the flow of control in a 'for' loop −
  - The **init** step is executed first, and only once. This step allows you to declare and initialize any number of loop control variables. You are not required to put a statement here, as long as a semicolon appears.
  - Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
  - After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
  - The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

for Loop



for(init; condition; increment)
{
conditional code ;
}
Init
condition
If condition is true
code block
increment
If condition is false

for Loop



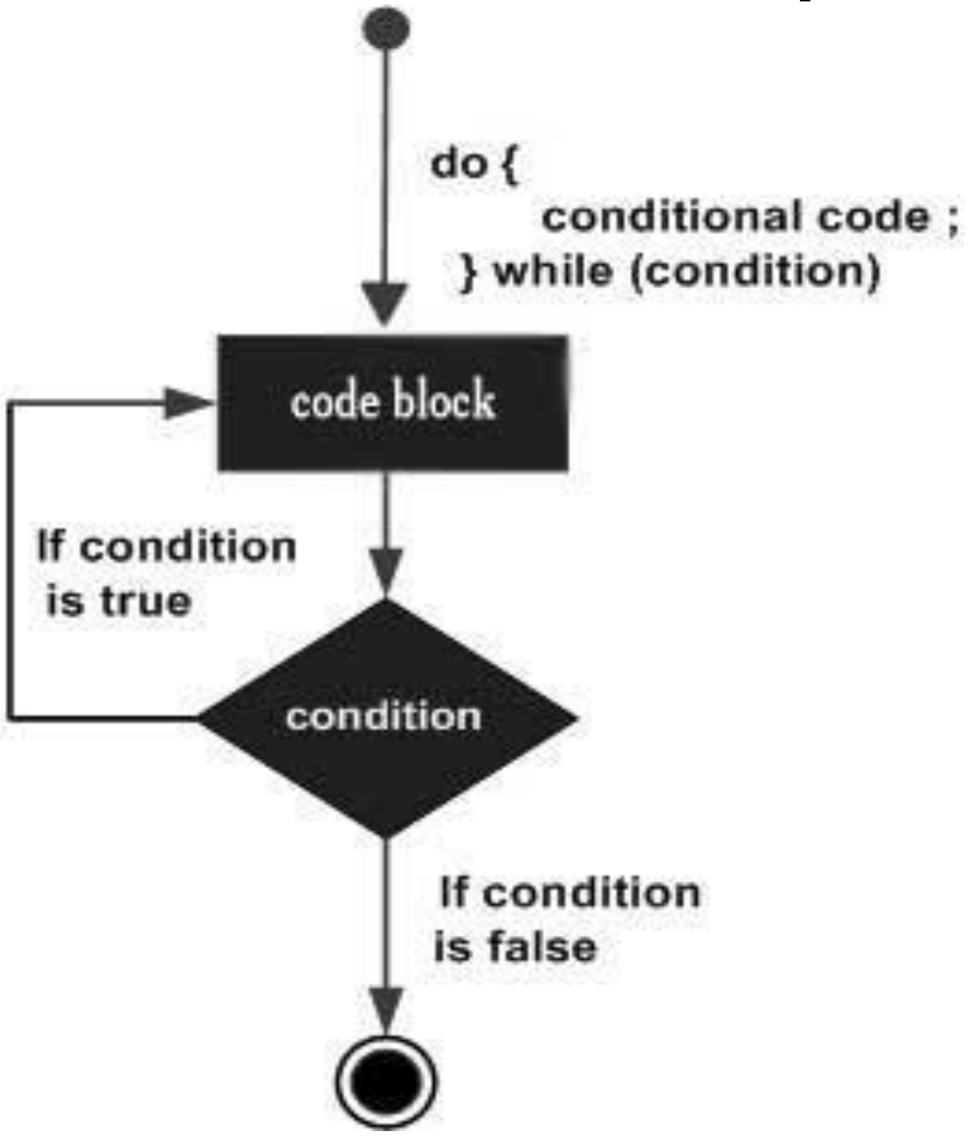
```
#include <stdio.h>

int main () {

    int a;

    /* for loop execution */
    for( a = 10; a < 20; a = a + 1 ){
       printf("value of a: %d\n", a);
    }

    return 0;
}
```

do…while Loop

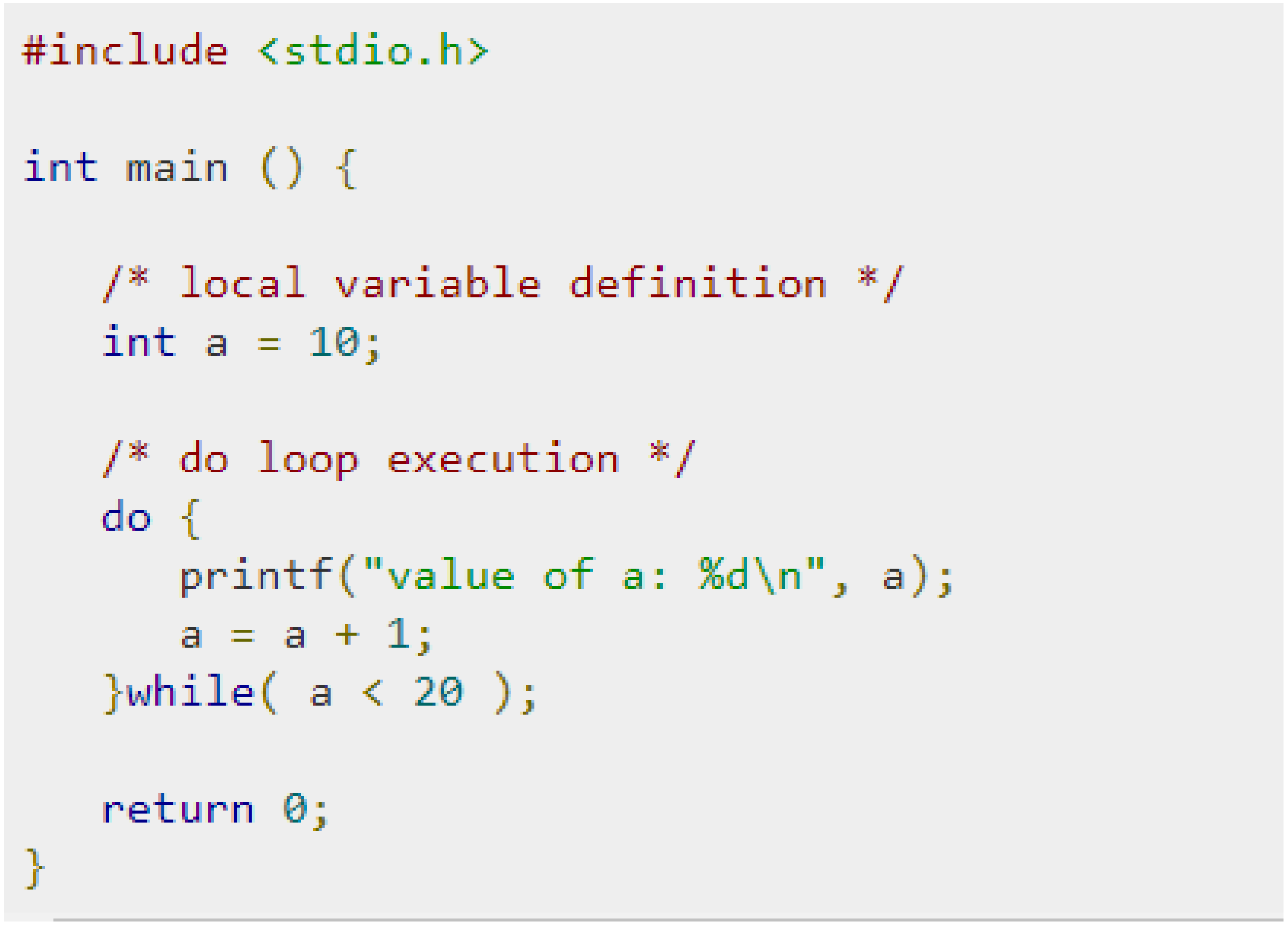
- Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.
- A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.
- The syntax of a **do...while** loop in C programming language is −

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

do…while Loop

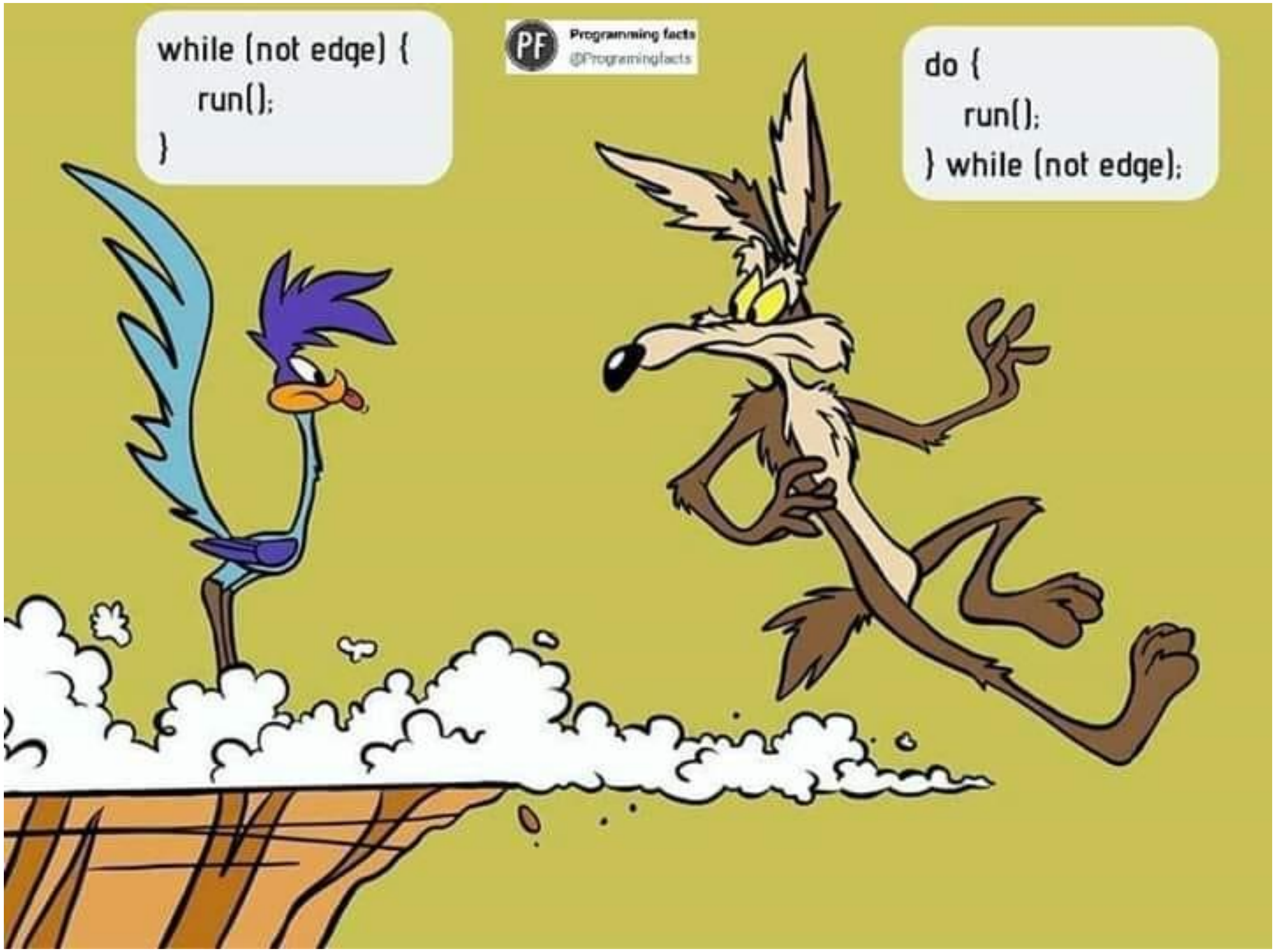


do…while Loop



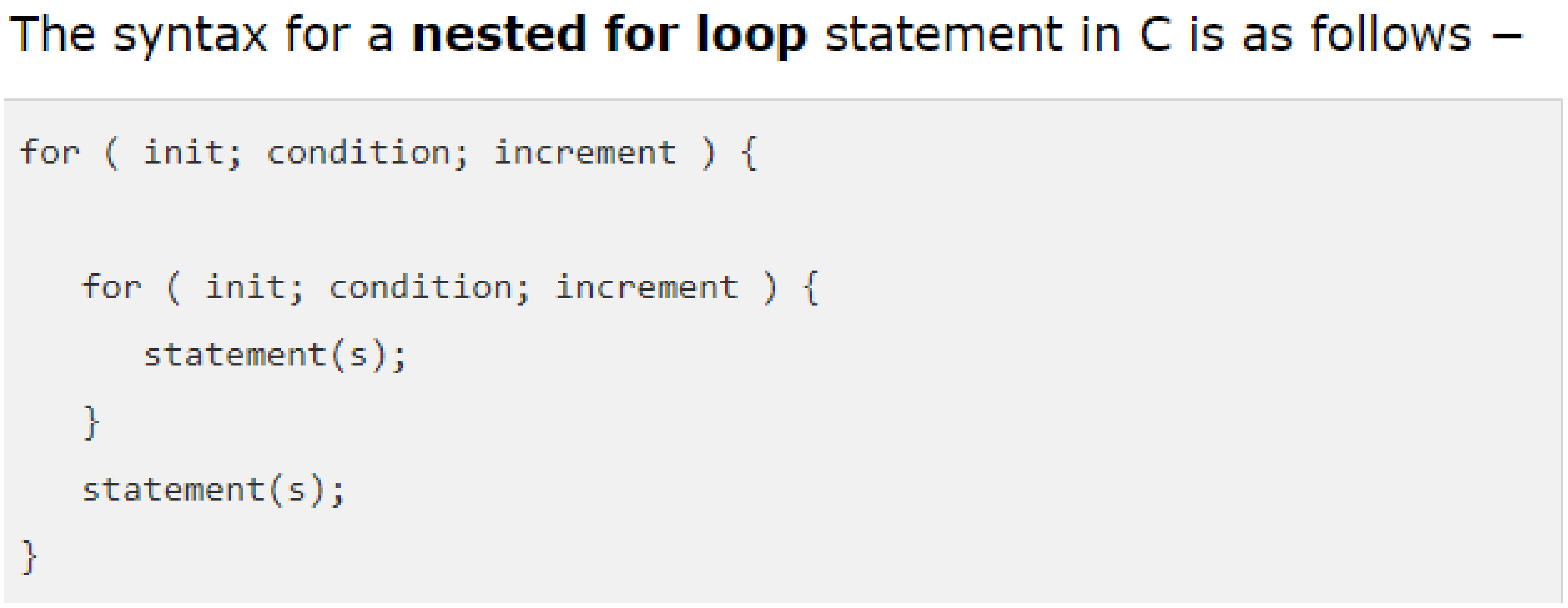
```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {
      printf("value of a: %d\n", a);
      a = a + 1;
   }while( a < 20 );

   return 0;
}
```

while vs do…while Loop



Nested loops in C

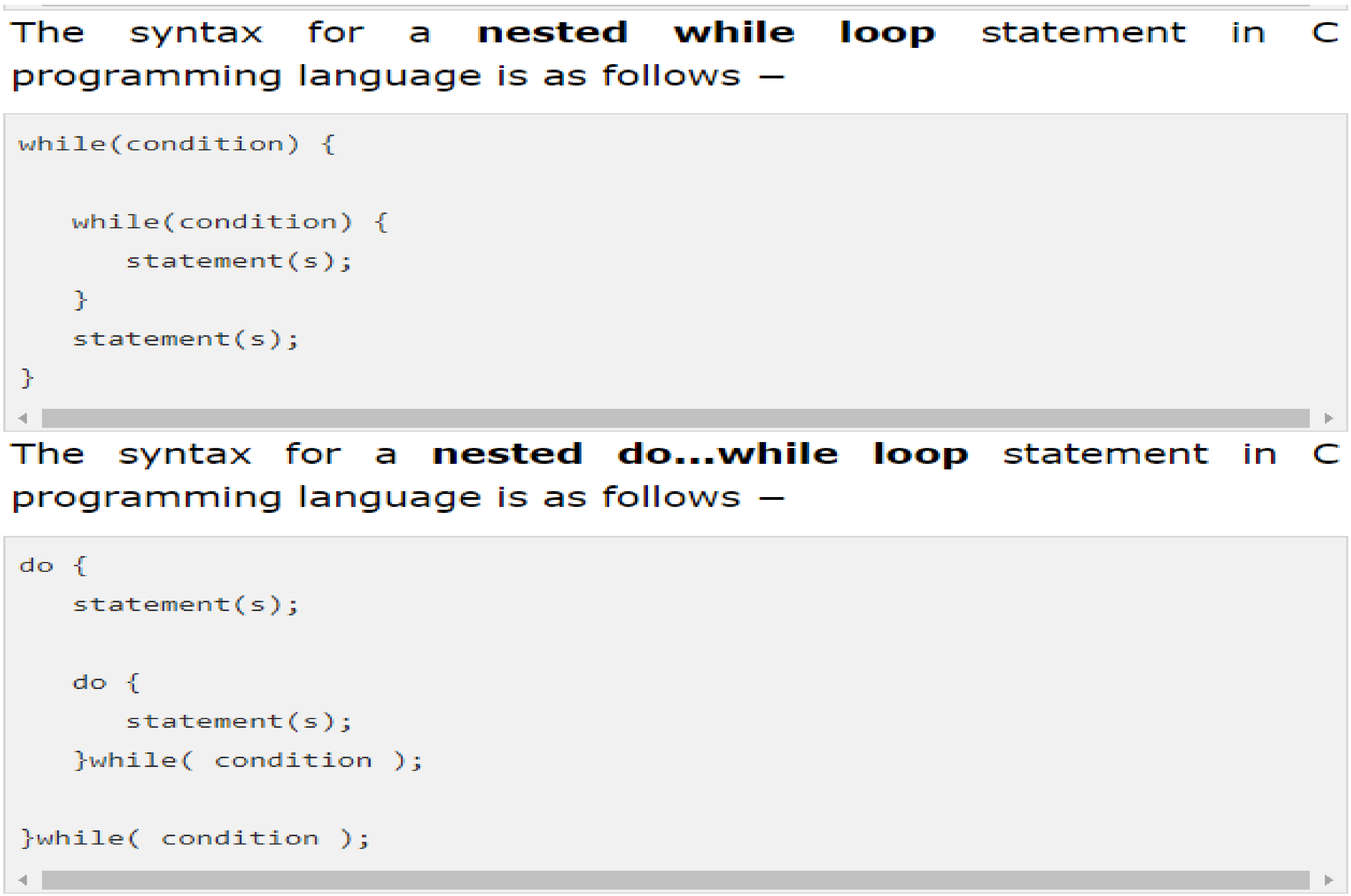
- C programming allows to use one loop inside another loop.



The syntax for a **nested for loop** statement in C is as follows −

```
for ( init; condition; increment ) {

   for ( init; condition; increment ) {
      statement(s);
   }
   statement(s);
}
```

Nested loops in C



The syntax for a **nested while loop** statement in C programming language is as follows −

```
while(condition) {

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

The syntax for a **nested do...while loop** statement in C programming language is as follows −

```
do {
   statement(s);

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

}while( condition );
```

Assignments 1

- Print the followings:

```
4 7
*******
*******
*******
*******
```

Assignments 2

- Print the followings:

```
4
   ****
  ****
 ****
****
```

Assignments 3

- Print the followings:

```
4
*
**
***
****
```