**UNIT 3**

**Control statements** 

**Control statements** enable us to specify the flow of program **control**; i.e., the order in which the instructions in a program must be executed.

**There are four types of control statements:**

1. **Sequence control statements**
2. **Decision control statements or conditional statements**
3. **Case control statements**
4. **Loop control statements**

**IF statement:**

It takes an expression in parenthesis and a statement or block of statements. if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped. Set of statements are written in {} (pair of braces).

**Syntax of IF statement:**

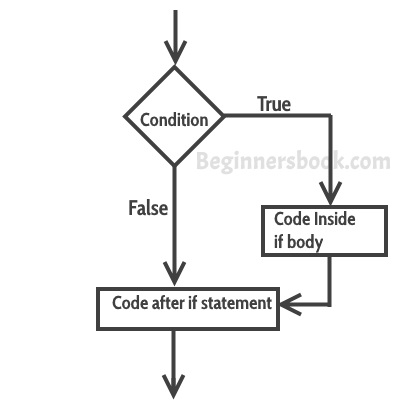
if (expression)

{

Block of statements;

}

**FLOWCHART of IF Statement**



**Example**

// Program to display a number if it is negative

#include <stdio.h>

int main () {

int number;

printf ("Enter an integer: ");

scanf ("%d", &number);

// true if number is less than 0

if (number < 0) {

printf ("You entered %d.\n", number);

}

printf ("The if statement is easy.");

return 0;

}

**Output 1**

Enter an integer: -2

You entered -2.

The if statement is easy.

When the user enters -2, the test expression number<0 is evaluated to true. Hence, You entered -2 is displayed on the screen.

**Output 2**

Enter an integer: 5

The if statement is easy.

When the user enters 5, the test expression number<0 is evaluated to false and the statement inside the body of if is not executed

**IF ELSE statement:**

The if...else statement executes two different codes depending upon whether the test expression is true or false.

If statement is true, then statements written in if clause will be executed otherwise statements written in else clause will be executed.

**Syntax of IF statement:**

if (expression)

{

Block of statements;

}

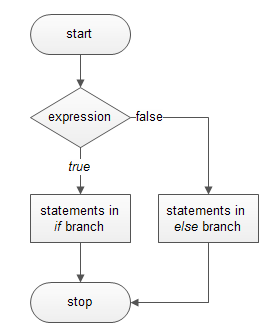
else

{

Block of statements;

}

**FLOWCHART of IF ELSE Statement**



**Example of if else statement**

Check whether an integer is odd or even

#include <stdio.h>

int main () {

int number;

printf ("Enter an integer: ");

scanf ("%d", &number);

// True if the remainder is 0

if (number%2 == 0) {

printf ("%d is an even integer.”, number);

}

else {

printf ("%d is an odd integer.”, number);

}

**Output**

Enter an integer: 7

7 is an odd integer.

When the user enters 7, the test expression number%2==0 is evaluated to false. Hence, the statement inside the body of else is executed.

**Nested if...else statement**

This is a special method to nest many control structures of similar types in one larger statement. The nesting can be done either in if statements or in else statements. The only thing to remember is that the statement starting later ends first and the structures starting earlier ends later.

**Syntax of Nested if...else statement**

if (expression)

{

Block of statements;

}

else

{

if(expression)

{

Block of statements;

}

}

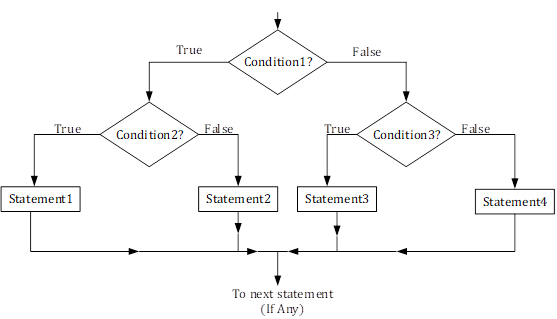
else

{

Block of statements;

}

**FLOWCHART of Nested IF ELSE Statement**



**Example of Nested if else statement**

#### **program to find largest from three numbers given by user to Explain Nested if-else**

**include**<stdio.h>

**int** main ()

{

**int** num1, num2, num3;

printf ("Enter three numbers:\n");

scanf ("%d%d%d”, &num1, &num2, &num3);

**if**(num1>num2)

{

/\* This is nested if-else \*/

**if**(num1>num3)

{

printf ("Largest = %d", num1);

}

**else**

{

printf ("Largest = %d", num3);

}

}

**else**

{

/\* This is nested if-else \*/

**if**(num2>num3)

{

printf ("Largest = %d", num2);

}

**else**

{

printf ("Largest = %d", num3);

}

}

**return** (0);

OUTPUT

Run 1:

---------------

Enter three numbers:

12↲

33↲

-17↲

Largest = 33

# Switch Statement

The switch statement allows us to execute one code block among many alternatives.

If value of expression matches with any of cases given, then that case will be executed. Switch and case two keywords used in switch case.one extra case called as default case is optional. The default case gets executed if no case satisfies the condition.

**Syntax of switch statement**

switch (expression)

​{

case constant1:

// statements

break;

case constant2:

// statements

break;

.

.

.

default:

// default statements

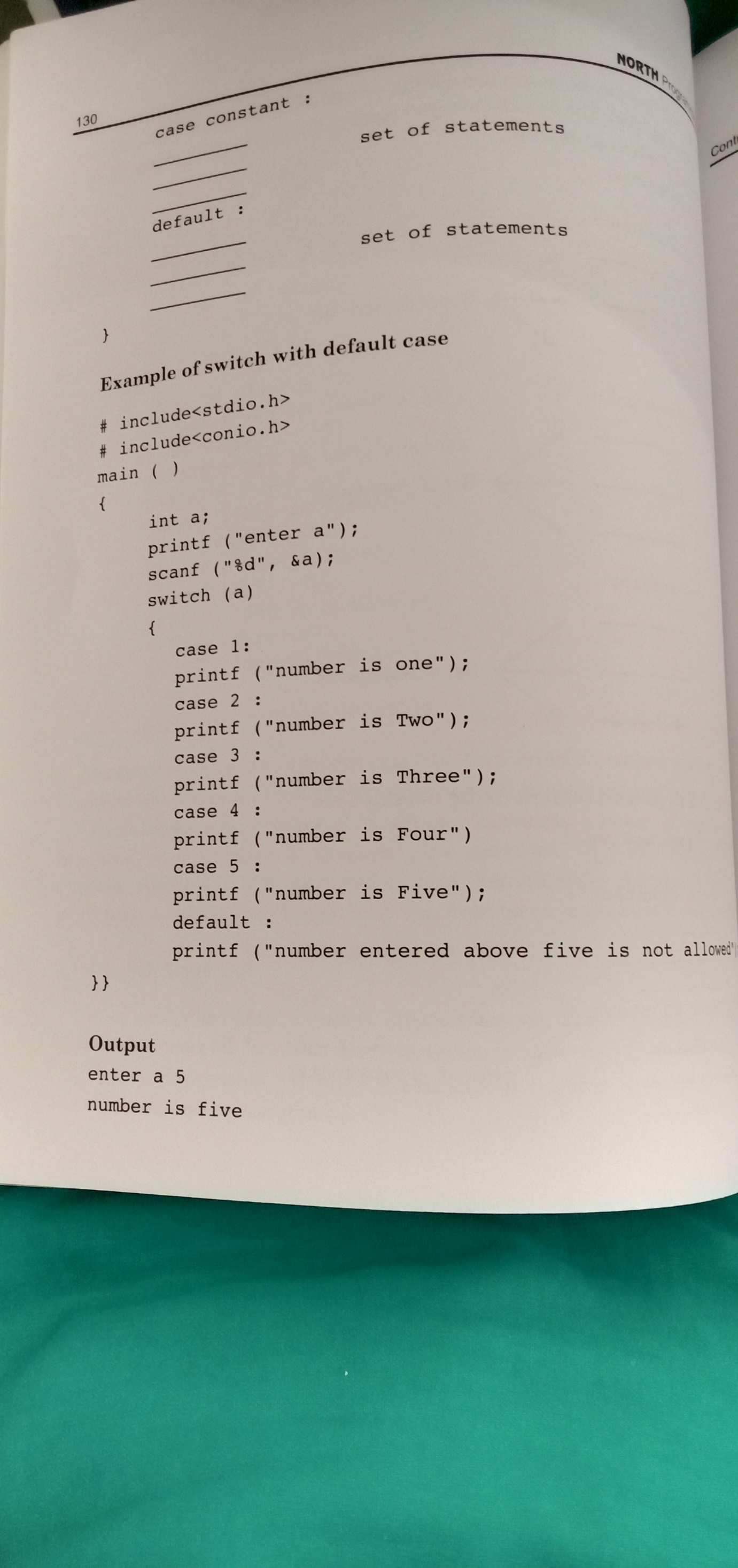
}

The expression is evaluated once and compared with the values of each case label.

* If there is a match, the corresponding statements after the matching label are executed. For example, if the value of the expression is equal to constant2, statements after case constant2: are executed until break is encountered.
* If there is no match, the default statements are executed.

If we do not use break, all statements after the matching label are executed.

By the way, the default clause inside the switch statement is optional.



**Break Statement**

Break statement is used to transfer control out of loop where it is defined. Due to break the first statement written outside loop gets executed. Break statements are useful when you want your program-flow to come out of the switch body. Whenever a break statement is encountered in the switch body, the control comes out of the switch case statement.

**Example of Switch Case with break**  
I’m taking the same above that we have seen above but this time we are using break.

#include <stdio.h>

int main ()

{

int i=2;

switch (i)

{

case 1:

printf ("Case1 ");

break;

case 2:

printf ("Case2 ");

break;

case 3:

printf ("Case3 ");

break;

case 4:

printf ("Case4 ");

break;

default:

printf ("Default ");

}

return 0;

}

**Output:**

Case 2

**LOOP**

A **Loop** executes the sequence of statements many times until the stated condition becomes false. A loop consists of two parts, a body of a loop and a control statement. The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false. The purpose of the loop is to repeat the same code a number of times.

'C' programming language provides us with three types of loop constructs:

1. The for loop

2. The while loop

3. The do-while loop

## For loop

A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop is as follows:

for (initial value; condition; incrementation or decrementation )

{

statements;

}

* The initial value of the for loop is performed only once.
* The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
* The incrementation/decrementation increases (or decreases) the counter by a set value.

#include<stdio.h>

int main ()

{

int number;

for (number=1; number<=10; number++) //for loop to print 1-10 numbers

{

printf ("%d\n”, number); //to print the number

}

return 0;

}

Output:

1

2

3

4

5

6

7

8

9

10

## While Loop

A while loop is the most straightforward looping structure. The basic format of while loop is as follows:

while (condition) {

statements;

}

It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop. If a condition is true, then and only then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false. Once the condition becomes false, the control goes out of the loop.

#include<stdio.h>

#include<conio.h>

int main ()

{

int num=1; //initializing the variable

while(num<=10) //while loop with condition

{

printf ("%d\n”, num);

num++; //incrementing operation

}

return 0;

}

Output:

1

2

3

4

5

6

7

8

9

10

## Do-While loop

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

The basic format of while loop is as follows:

do {

statements

} while (expression);

As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop.

In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

#include<stdio.h>

#include<conio.h>

int main ()

{

int num=1; //initializing the variable

do //do-while loop

{

printf("%d\n",2\*num);

num++; //incrementing operation

} while(num<=10);

return 0;

}

Output:

2

4

6

8

10

12

14

16

18

20

**Continue Statement**

The **continue statement** is used inside [loops](https://beginnersbook.com/2014/01/c-loops-examples/). When a continue statement is encountered inside a loop, control jumps to the beginning of the loop for next iteration, skipping the execution of statements inside the body of loop for the current iteration.

The **continue statement in C** programming works somewhat like the break **statement**. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between. For the for loop, **continue statement** causes the conditional test and increment portions of the loop to execute.

**Example: continue statement inside for loop**

#include <stdio.h>

int main ()

{

for (int j=0; j<=8; j++)

{

If (j==4)

{

/\* the continue statement is encountered when

\* the value of j is equal to 4.

\*/

Continue;

}

/\* this print statement would not execute for the

\* loop iteration where j ==4 because in that case

\* This statement would be skipped.

\*/

Printf ("%d ", j);

}

return 0;

}

Output:

0 1 2 3 5 6 7 8

Value 4 is missing in the output, why? When the value of variable j is 4, the program encountered a continue statement, which makes the control to jump at the beginning of the for loop for next iteration, skipping the statements for current iteration (that’s the reason printf didn’t execute when j is equal to 4).

**Goto Statement**

The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

### Syntax of goto Statement

goto label;

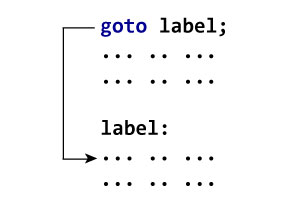
... .. ...

... .. ...

label:

statement;

The label is an identifier. When the goto statement is encountered, the control of the program jumps to label: and starts executing the code.



|  |
| --- |
| // C program to check if a number is  // even or not using goto statement  #include <stdio.h>    // function to check even or not  void checkEvenOrNot(int num)  {      if (num % 2 == 0)          // jump to even          goto even;      else          // jump to odd          goto odd;    even:      printf("%d is even", num);      // return if even      return;  odd:      printf("%d is odd", num);  }    int main() {      int num = 26;      checkEvenOrNot(num);      return 0;  } |

**Output:**

26 is even

**Exit () Function**

Exit is predefined function. It is written in stdlib.h header File. It can terminate or finish execution of program.

## Example

The following example shows the usage of exit () function.

[Live Demo](http://tpcg.io/CxuyMt)

#include <stdio.h>

#include <stdlib.h>

int main () {

printf ("Start of the program....\n");

printf ("Exiting the program....\n");

exit (0);

printf ("End of the program....\n");

return(0);

}

Let us compile and run the above program that will produce the following result −

Start of the program....

Exiting the program....