Unconditional Statements:

In C, an unconditional statement is a statement that is executed without any condition or consideration of whether a particular condition is true or false. Unconditional statements are executed sequentially, one after the other, and they do not depend on any conditional logic.

Here are some common examples of unconditional statements in C:

1. Goto
2. Break
3. Continue

The “goto” Statement :

- C supports the “goto” statement to branch unconditionally from one point to another in the program.
- Although it may not be essential to use the “goto” statement in a highly structured language like “C”, there may be occasions when the use of goto is necessary.
- The goto requires a label in order to identify the place where the branch is to be made.
- A label is any valid variable name and must be followed by a colon (:).
- The label is placed immediately before the statement where the control is to be transferred.
- The label can be anywhere in the program either before or after the goto label statement.

Syntax :

```
goto label1;

............
label:
statement;
```

In this syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.

Forward Jump	**Backward Jump**
goto label;	label:
....	Statement;
label:	....
statement;	goto label;

- During running of a program, when a statement likes “goto begin;” is met, the flow of control will jump to the statement immediately following the label “begin:” this happens unconditionally.
- “goto” breaks the normal sequential execution of the program.
- If the “label:” is before the statement “goto label;” a loop will be formed and some statements will be executed repeatedly. Such a jump is known as a “**backward jump**”.
- If the “label:” is placed after the “goto label;” some statements will be skipped and the jump is known as a “**forward jump**”.

Example:

```
# include<stdio.h>
void main( )
  {
    int  i=1, num, sum =0;
    for(i=0; i < 5; i ++)
     {
      printf(“ Enter an integer:”);
      scanf( “%d”, &num);
      if(num < 0)
     {
        printf(“\nyou have entered a negative number”);
        continue ;     /* skip the remaining part of loop */
    }
    sum += num;
  }
printf(“The sum of the Positive Integers Entered = % d \ n”, sum);
}
```

The “break” Statement :

A break statement terminates the execution of the loop and the control is transferred to the statement immediately following the loop. i.e., the break statement is used to terminate loops or to exit from a switch.

- It can be used within for, while, do-while, or switch statement.
- The break statement is written simply as break;
- In case of inner loops, it terminates the control of inner loop only.

Syntax :

```
        Jump-statement;
         break;
```

The jump statement in c break syntax can be while loop, do while loop, for loop or switch case.

Example:

```
#include<stdio.h>
main( ){
  int i;
  for (i=1; i<=5; i++){
    printf ("%d", i);
    if (i==3)
    break;
  }
}
```

The “continue” Statement :

- The continue statement is used to bypass the remainder of the current pass through a loop.
- The loop does not terminate when a continue statement is encountered. Instead, the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop.
- The continue statement can be included within a while, do-while, for statement.
- It is simply written as “continue”.
- The continue statement tells the compiler “Skip the following Statements and continue with the next Iteration”.
- In “while” and “do” loops continue causes the control to go directly to the test –condition and then to continue the iteration process.
- In the case of “for” loop, the updation section of the loop is executed before test-condition, is evaluated.

Syntax :

```
        Jump-statement;
        Continue;
```

The jump statement can be while, do while and for loop.

Example:

```
# include<stdio.h>
void main( )
  {
    int  i=1, num, sum =0;
    for(i=0; i < 5; i ++)
     {
      printf(“ Enter an integer:”);
      scanf( “%d”, &num);
      if(num < 0)
     {
        printf(“\nyou have entered a negative number”);
        continue ;     /* skip the remaining part of loop */
   }
    sum += num;
  }
printf(“The sum of the Positive Integers Entered = % d \ n”, sum);
}
```