Difference between Formatted and Unformatted Functions

Sr No.	Formatted I/O Function	Unformatted I/O Function
1	Formatted I/O functions allow to supply input or display output in user desired format.	Unformatted I/O functions are the most basic form of input and output and they do not allow to supply input or display output in user desired format.
2	printf() and scanf() are examples for formatted input and output functions.	getch(), getche(), getchar(), gets(), puts(), putchar() etc. are examples of unformatted input output functions.
3	Formatted input and output functions contain format specifier in their syntax.	Unformatted input and output functions do not contain format specifier in their syntax.
4	Formatted I/O functions are used for storing data more user friendly.	Unformatted I/O functions are used for storing data more compactly.
5	Formatted I/O functions are used with all data types.	Unformatted I/O functions are used mainly for character and string data types.

Formatted I/O Example:

```
#include<stdio.h>
#include<conio.h>

void main()
{
 int a;
 clrscr();
 printf(“Enter value of a:”);
 scanf(“%d”, &a);
 printf(“ a = %d”, a);
 getch();
}
```

Output :

```
Enter value of a:5↲
a = 5
```

Unformatted I/O Example:

```
#include<stdio.h>
#include<conio.h>

void main()
{
 char ch ;
 clrscr();
 printf(“Press any character:”);
 ch = getche();
 printf(“\nYou pressed :”
 putchar(ch);
getch();
}
```

Output :

```
Press any character: L
You pressed: L
```

Pre-Increment Operator in C

As the name suggests, the pre-increment operator alters the value of the variable before using it in any expression. Therefore, we can say that the pre-increment operator increases the value of the variable first and then use it in the expression.

Syntax:

b = ++a

For example, if the initial value of a were 5, then the value 6 would be assigned to b.

```
#include<stdio.h>
// Main function
int main(){
	// Declare a variable (say a) and assign 5 to it.
	int a = 5;

	// Now use pre-increment on 'a' and assign it to 'b'.
	int b = ++a;

	// Print 'a' and 'b'
	printf("a = %d\nb = %d\n", a, b);

	return 0;
}
```

Output:

```
a = 6
b = 6
```

Post-Increment Operator in C

The post-increment operator is used when it is required to increment the value of the variable after evaluating the expression. Therefore, in post-increment value is first used in the expression, and then it is incremented.

Syntax:

b = a++;

For example, assume the initial value of a to be 5. Then after executing the above statement the final value of b will be 5 as the value of a will be incremented after performing the expression.

```
#include<stdio.h>

// Main function
int main(){
	// Declare a variable (say a) and assign 5 to it.
	int a = 5;

	// Now use post-increment on 'a' and assign it to 'b'.
	int b = a++;

	// Print 'a' and 'b'
	printf("a = %d\nb = %d\n", a, b);

	return 0;
}
```

Output:

```
a = 6
b = 5
```

Difference between While and Do While Loop:

while	**do-while**
Condition is checked first then statement(s) is executed.	Statement(s) is executed at-least once, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
No semicolon at the end of while. while(condition)	**Semicolon at the end of while. while(condition);**
If there is a single statement, brackets are not required.	Brackets are always required.
Variable in condition is initialized before the execution of loop.	variable may be initialized before or within the loop.
while loop is entry controlled loop.	do-while loop is exit controlled loop.
Syntax of while loop: while (condition) { Block of statements; } Statement-x;	**Syntax of do-while loop:** do { Block of statements; } while (condition); Statement-x;
Program of while loop: #include <stdio.h> Void main() { int i; i = 1; while(i<=10) { printf("hello"); i = i + 1; } }	**Program of do-while loop:** #include <stdio.h> Void main() { int i; i = 1; do { printf("hello"); i = i + 1; } while(i<=10); }

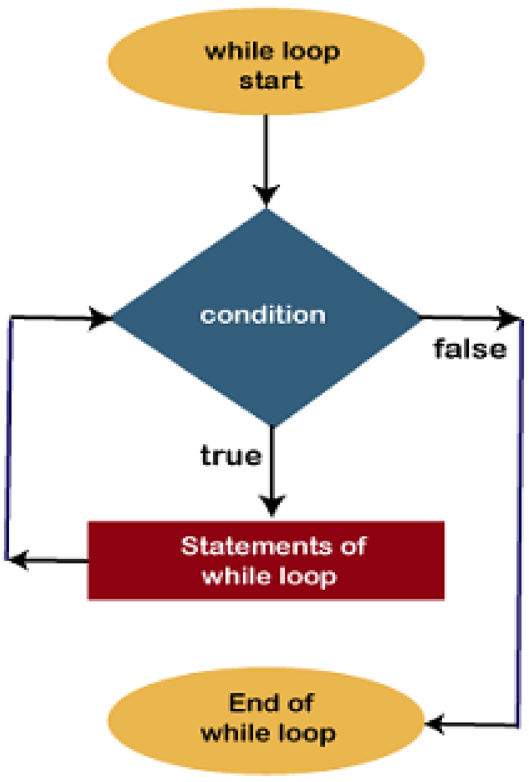
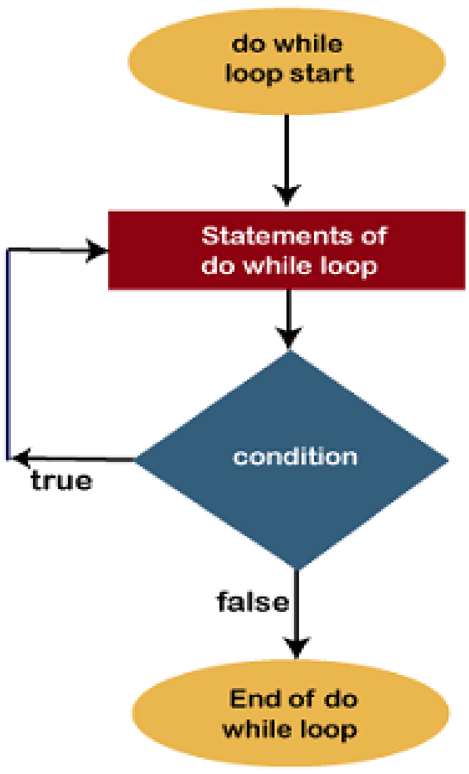
Syntax of while loop:
```
while (condition)
{
    Block of statements;
}
Statement-x;
```

Syntax of do-while loop:
```
do
{
    Block of statements;
}
while (condition);
Statement-x;
```

Program of while loop:
```
#include <stdio.h>
Void main()
{
   int i;
   i = 1;
  while(i<=10)
  {
   printf("hello");
   i = i + 1;
  }
}
```

Program of do-while loop:
```
#include <stdio.h>
Void main()
{
   int i;
   i = 1;
  do
  {
   printf("hello");
   i = i + 1;
  }
 while(i<=10);

}
```

while	do-while
Flowchart of while loop:  while loop start condition false true Statements of while loop End of while loop	**Flowchart of do-while loop:**  do while loop start Statements of do while loop condition true false End of do while loop

Difference Between Break and Continue Statement in C

Break Statement	Continue Statement
The Break statement is used to exit from the loop constructs.	The continue statement is not used to exit from the loop constructs.
The break statement is usually used with the switch statement, and it can also use it within the while loop, do-while loop, or the for-loop.	The continue statement is not used with the switch statement, but it can be used within the while loop, do-while loop, or for-loop.
When a break statement is encountered then the control is exited from the loop construct immediately.	When the continue statement is encountered then the control automatically passed from the beginning of the loop statement.
Syntax: break;	**Syntax:** continue;
Break statements uses switch and label statements.	It does not use switch and label statements.
Leftover iterations are not executed after the break statement.	Leftover iterations can be executed even if the continue keyword appears in a loop.

Comparison Chart Between Global Variable and Local Variable

Global Variable	Local Variable
Global variables are declared outside all the function blocks.	Local Variables are declared within a function block.
The scope remains throughout the program.	The scope is limited and remains within the function only in which they are declared.
Any change in global variable affects the whole program, wherever it is being used.	Any change in the local variable does not affect other functions of the program.
A global variable exists in the program for the entire time the program is executed.	A local variable is created when the function is executed, and once the execution is finished, the variable is destroyed.
It can be accessed throughout the program by all the functions present in the program.	It can only be accessed by the function statements in which it is declared and not by the other functions.
If the global variable is not initialized, it takes zero by default.	If the local variable is not initialized, it takes the garbage value by default.
Global variables are stored in the data segment of memory.	Local variables are stored in a stack in memory.
We cannot declare many variables with the same name.	We can declare various variables with the same name but in other functions.