Formatted & Unformatted Input Output in C Programming

In C programming, input means taking data from the user and output means displaying data to the user. C provides two types of input/output functions :

1. Formatted Input/Output

- Formatted I/O functions use a format specifier to specify the type and format of data.
- These functions give more control over how the data is read or displayed.
- Commonly used functions are printf() and scanf().
- Format specifiers like %d, %f, %c, %s are used.

Examples :

1. Formatted Output using printf()

```
#include <stdio.h>
int main() {
    int age = 20;
    float height = 5:8;
    printf("Age = %d\n", age);
    printf("Height = %.1f\n", height);
    return 0;
}
```

2. Formatted Input using scanf()

```
#include <stdio.h>
int main() {
    int age;
    printf("Enter your age:");
    scanf("%d", &age);
    printf("You entered: %d\n", age);
    return 0;
}
```

2. Unformatted Input/Output

- Unformatted I/O functions do not use any format specifier.
- These functions read or write data as it is (character by character or string by string).
- They are generally used for character and string input/output.
- Commonly used functions are getchar(), putchar(), gets(), puts().
- They are simpler but provide less control compared to formatted I/O.

Examples :

1. Unformatted Output using putchar() and puts()

```
#include <stdio.h>
int main() {
   putchar('A');                  // prints single character
   puts("Hello C Programming"); // prints string
   return 0;
}
```

2. Unformatted Input using getchar() and gets()

```
#include <stdio.h>
int main() {
   char ch;
   char name[50];
   ch = getchar();       // reads a single character
   gets(name);       // reads a string (not recommended
                        in modern C, use fgets())
   printf("You entered: %c\n", ch);
   printf("Name: %s\n", name);
   return 0;
}
```

Summary : Formatted I/O is more flexible and commonly used for general purpose input/output. Unformatted I/O is useful for simple character and string handling.

While, Do-While and For Statements in C Programming

Loop statements are used to repeat a block of code multiple times in C language. C provides three types of loop statements : while, do-while and for.

1. While Statement

- The while statement is an entry controlled loop, which means the condition is checked before executing the loop body.
- If the condition is true, the statements inside the loop are executed. If the condition is false, the loop is terminated.
- Syntax :

```
while (condition)
{
    // statements;
}
```

Example : :

```
#include <stdio.h>
int main() {
    int i = 1;
    while (i <= 5) {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

Output : 1 2 3 4 5

2. Do-While Statement

- The do-while statement is an exit controlled loop, which means the statements inside the loop are executed at least once, and then the condition is checked.
- If the condition is true, the loop continues to execute. If the condition is false, the loop is terminated.
- Syntax :

```
do
{
    // statements;
} while (condition);
```

Example :

```
#include <stdio.h>
int main() {
    int i = 1;
    do {
        printf("%d\n", i);
        i++;
    } while (i <= 5);
    return 0;
}
```

Output : 1 2 3 4 5

3. For Statement

- The for statement is a counted loop, which is used when the number of iterations is known.
- It consists of three parts : initialization, condition and increment/decrement.
- Syntax :

```
for (initialization; condition; increment)
{
    // statements;
}
```

Example : :

```
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

Output : 1 2 3 4 5

Conclusion : All three loops (while, do-while and for) are used to repeat a block of code, but they differ in the way the condition is checked and how the loop is executed.

Nested Loops, If-Else and Switch in C Programming

In C programming, nested loops, if-else and switch are important control structures that help in making decisions and repeating tasks in a program.

1. Nested Loops

- Nested loops are loops inside another loop.
- They are used when we need to repeat a set of instructions multiple times in a structured way (e.g., printing patterns, working with matrices).
- The outer loop controls the number of times the inner loop will run.

Example :

Print a pattern using nested loops.

```
#include <stdio.h>
int main() {
    int i, j;
    for (i = 1; i <= 4; i++) {
        for (j = 1; j <= i; j++) {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

Output :

```
*
**
***
****
```

2. If-Else

- The if-else statement is used to check a condition and execute different blocks of code based on whether the condition is true or false.
- It helps in decision making in a program.

Syntax :

```
if (condition) {
    // statements if true
} else {
    // statements if false
}
```

Example :

Check if a number is even or odd.

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num % 2 == 0) {
        printf("Even number");
    } else {
        printf("Odd number");
    }
    return 0;
}
```

Output :

```
Enter a number: 6
Even number
```

3. Switch

- The switch statement is used to select one of many blocks of code to be executed based on the value of a variable or expression.
- It is useful when we have multiple conditions to check for the same variable.

Syntax :

```
switch (expression) {
    case value1:
        // statements
        break;
    case value2:
        // statements
        break;
    ...
    default:
        // statements
}
```

Example :

Find the day of the week using switch.

```
#include <stdio.h>
int main() {
    int day;
    printf("Enter day number (1-7): ");
    scanf("%d", &day);
    switch (day) {
        case 1: printf("Monday"); break;
        case 2: printf("Tuesday"); break;
        case 3: printf("Wednesday"); break;
        case 4: printf("Thursday"); break;
        case 5: printf("Friday"); break;
        case 6: printf("Saturday"); break;
        case 7: printf("Sunday"); break;
        default: printf("Invalid day");
    }
    return 0;
}
```

Output :

```
Enter day number (1-7): 3
Wednesday
```

Conclusion : Nested loops are used for repeated tasks, if-else helps in decision making, and switch provides a cleaner way to handle multiple choices in C programming.

Break - Continue Statements in C Programming

In C programming, break and continue are control statements that are used inside loops (for, while, do-while) to change the normal flow of execution.

1. Break Statement

- The break statement is used to exit from the loop immediately.
- It terminates the loop and control moves to the statement after the loop.
- It is mainly used when a specific condition is met and we want to stop the loop.

Syntax :

```
break;
```

Example :

Print numbers from 1 to 10, but stop when the number is 5.

```
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i <= 10; i++) {
        if (i == 5) {
            break;  // exit the loop when i = 5
        }
        printf("%d ", i);
    }
    return 0;
}
```

Output : 1 2 3 4

2. Continue Statement

- The continue statement is used to skip the current iteration of the loop.
- It does not terminate the loop, but moves the control to the next iteration.
- It is useful when we want to ignore a particular value and continue with the next iteration.

Syntax :

```
continue;
```

Example :

Print numbers from 1 to 10, but skip 5.

```
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i <= 10; i++) {
        if (i == 5) {
            continue;  // skip 5 and continue
        }
        printf("%d ", i);
    }
    return 0;
}
```

Output : 1 2 3 4 6 7 8 9 10

Key Differences :

Feature	Break Statement	Continue Statement
Purpose	Terminates the loop completely	Skips the current iteration
Control moves to	After the loop	Next iteration of the loop
Use case	When we want to stop the loop	When we want to ignore a value and continue

Conclusion : Break and continue statements help us control the flow of loops in C programming. Break exits the loop, while continue skips the current iteration and continues with the next one.