CONDITIONAL STATEMENTS

Certainly! Conditional statements in C are used to control the flow of a program based on certain conditions. The primary conditional statements in C are `if`, `else if`, and `else`. Here's an explanation in text format followed by code examples:

`if` Statement:

The `if` statement is used to execute a block of code only if a specified condition is true.

Syntax:

```c
if (condition) {
    // code to be executed if the condition is true
}
```

Example:

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

```
int main() {
    int x = 10;

    if (x > 5) {
        printf("x is greater than 5\n");
    }

    return 0;
}
```

`if-else` Statement:

The `if-else` statement is used when you want to execute one block of code if the condition is true and another block if the condition is false.

Syntax:

```c
if (condition) {
    // code to be executed if the condition is true
} else {
    // code to be executed if the condition is false
```

}
```

Example

```c
#include <stdio.h>

int main() {
    int x = 3;

    if (x > 5) {
        printf("x is greater than 5\n");
    } else {
        printf("x is not greater than 5\n");
    }

    return 0;
}
```
```

`if-else if-else` Statement:

The `if-else if-else` statement is used when there are multiple conditions to be checked in a sequential manner.

Syntax:

```c
if (condition1) {
    // code to be executed if condition1 is true
} else if (condition2) {
    // code to be executed if condition2 is true
} else {
    // code to be executed if none of the conditions is true
}
```

Example:

```c
#include <stdio.h>

int main() {
    int x = 7;
```

```
    if (x > 10) {
        printf("x is greater than 10\n");
    } else if (x > 5) {
        printf("x is greater than 5 but not greater than 10\n");
    } else {
        printf("x is not greater than 5\n");
    }

    return 0;
}
```

These conditional statements allow you to control the flow of your program based on different conditions.

ELSE-IF:

The else if statement in C is used when you have multiple conditions to check in a sequential manner, and you want to execute the block of code associated with the first true condition. It extends the functionality of the if and else statements.

Syntax:

```
if (condition1) {
    // code to be executed if condition1 is true
} else if (condition2) {
    // code to be executed if condition2 is true
} else {
    // code to be executed if none of the conditions is true
}
```

Explanation:

The if block is executed if condition1 is true.

If condition1 is false, it moves to the else if block. Here, condition2 is checked, and if it is true, the corresponding block of code is executed.

If neither condition1 nor condition2 is true, the else block (if present) is executed.

Example:

c

```
#include <stdio.h>

int main() {
    int x = 7;
```

```
    if (x > 10) {
        printf("x is greater than 10\n");
    } else if (x > 5) {
        printf("x is greater than 5 but not greater than 10\n");
    } else {
        printf("x is not greater than 5\n");
    }

    return 0;
}
```

SHORT-HAND if:

The shorthand if statement, also known as the ternary operator, provides a concise way to write simple conditional expressions in a single line. It's a condensed form of the standard if-else statement. The syntax of the shorthand if statement is as follows:

c

condition ? expression_if_true : expression_if_false;

If the condition is true, the expression_if_true is evaluated.

If the condition is false, the expression_if_false is evaluated.

Example:

c

```
#include <stdio.h>

int main() {
    int x = 10;
    int result;

    // Shorthand if
    result = (x > 5) ? 1 : 0;

    printf("Result: %d\n", result);

    return 0;
}
```

In this example, if x is greater than 5, result will be set to 1; otherwise, it will be set to 0. The shorthand if statement is useful for simple conditional assignments and can make the code more compact and readable in certain situations.

It's important to note that while the ternary operator can enhance code readability for concise conditions, it's not always the best choice for complex conditions or when the expressions involved are too complex.

In such cases, using a regular if-else statement is usually more appropriate.

Switch:

The switch statement in C is used to select one of many code blocks to be executed. It provides an alternative to a series of if-else if-else statements when you need to compare a variable against multiple values.

Syntax:

c

```
switch (expression) {
    case constant1:
        // code to be executed if expression equals constant1;
        break;
    case constant2:
        // code to be executed if expression equals constant2;
        break;
    // additional cases as needed
    default:
        // code to be executed if none of the cases match expression;
```

}

The switch statement evaluates the expression and compares it with the values specified in the case statements.

If a match is found, the code block associated with that case is executed until a break statement is encountered.

If no match is found, the code block following the default label is executed.

Example:

c

```
#include <stdio.h>


int main() {
    char grade = 'B';


    switch (grade) {
        case 'A':
            printf("Excellent!\n");
            break;
        case 'B':
            printf("Good job!\n");
            break;
```

```
            case 'C':
                printf("Passing grade.\n");
                break;
            case 'D':
                printf("Barely passed.\n");
                break;
            case 'F':
                printf("Failed.\n");
                break;
            default:
                printf("Invalid grade.\n");
        }

        return 0;
}
```

In this example, the switch statement is used to determine the message to be printed based on the value of the variable grade. The break statements are important; they exit the switch block once the corresponding code block is executed.

It's worth noting that the switch statement only works with integral types (char, int, etc.) and can't be used directly with strings.

Additionally, cases cannot have ranges, and each case must end with a break statement unless you intentionally want the execution to "fall through" to the next case.