◇ Overview of Conditional (Branching) Statements in C++

Branching statements allow the program to make decisions and alter the flow of execution based on specific conditions.

Common Conditional Constructs:

- `if`, `if-else`, `if-else-if`
- `switch`
- Ternary operator (`?:`)
- `goto` (rarely used; not recommended)

◇ 2. `if`, `if-else`, and `if-else-if` Statements

Syntax:

```
if (condition) {
    // code block
} else if (another_condition) {
    // another code block
} else {
    // default code block
}
```

Example:

```
#include <iostream>
using namespace std;

int main() {
    int x = 20;

    if (x > 25) {
        cout << "Greater than 25\n";
    } else if (x == 20) {
        cout << "Exactly 20\n";
    } else {
        cout << "Less than 20\n";
    }

    return 0;
}
```

---

3. Ternary Operator (?:) – Conditional Expression

Syntax:

```
(condition) ? expression_if_true : expression_if_false;
```

Example:

```
#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 20;

    string result = (a > b) ? "A is greater" : "B is greater or equal";
    cout << result << endl;

    return 0;
}
```

---

4. switch Statement – Multi-Way Branching

Syntax:

```
switch (expression) {
    case constant1:
        // statements
        break;
    case constant2:
        // statements
        break;
    default:
        // default statements
}
```

Example:

```
#include <iostream>
using namespace std;

int main() {
    int day = 3;

    switch (day) {
        case 1: cout << "Monday\n"; break;
        case 2: cout << "Tuesday\n"; break;
        case 3: cout << "Wednesday\n"; break;
        default: cout << "Invalid Day\n";
    }
```

```
    return 0;
}
```

◇ 5. Enhancements in C++11, C++14, and C++17

☑ C++11: Scoped Enums (`enum class`) with `switch`

- Prevents naming collisions.
- Forces qualification of enum values.

Example:

```
#include <iostream>
using namespace std;

enum class Color { Red, Green, Blue };

int main() {
    Color c = Color::Green;

    switch (c) {
        case Color::Red: cout << "Red\n"; break;
        case Color::Green: cout << "Green\n"; break;
        case Color::Blue: cout << "Blue\n"; break;
        // No default needed if all enum values are handled
    }

    return 0;
}
```

☑ C++14: Generic Lambdas in Conditions

- Generic lambdas simplify condition logic when used inline with branching.

Example:

```
#include <iostream>
using namespace std;

int main() {
    auto check = [](auto x) {
        return (x % 2 == 0) ? "Even" : "Odd";
    };

    cout << check(10) << endl;
    cout << check(7.5) << endl;
```

```
    return 0;
}
```

☑ C++17: `if constexpr` – Compile-Time Conditional Execution

Used primarily in templates to allow compile-time branching.

▧ Syntax:

```
if constexpr (condition) {
    // evaluated at compile time
}
```

Example:

```
#include <iostream>
using namespace std;

template <typename T>
void printType(T val) {
    if constexpr (is_integral<T>::value) {
        cout << val << " is an integer\n";
    } else {
        cout << val << " is NOT an integer\n";
    }
}

int main() {
    printType(42);
    printType(3.14);
    return 0;
}
```

🔍 Why `if constexpr`?

- Eliminates unreachable branches at compile time.
- Useful in generic code to avoid invalid template instantiations.

◇ 6. Nested Conditional Statements

☑ Example:

```
#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 20;
```

```
    if (a < b) {
        if (b - a > 5)
            cout << "b is much larger than a\n";
        else
            cout << "b is slightly larger than a\n";
    }

    return 0;
}
```

◇ 7. Best Practices

Tip	Explanation
Avoid deep nesting	Use functions or early returns
Use `switch` with `enum class`	Safer and more readable
Prefer `if constexpr` in templates	For compile-time checks
Ternary operator for simple cases	Don't overuse for complex logic
Don't forget `break` in `switch`	Avoid fall-through unless intentional

---

◇ 8. Visual Comparison (C++11–C++17 Features)

Feature	C++98	C++11	C++14	C++17
`if`, `else`, `switch`	✅	✅	✅	✅
`enum class` in `switch`	❌	✅	✅	✅
Generic lambdas in condition	❌	❌	✅	✅
`if constexpr`	❌	❌	❌	✅
