在 C++ 中，**分支结构**和**循环结构**是控制程序流的两大基本结构。它们用于根据条件执行不同的代码块，以及在满足特定条件时重复执行代码。下面是对这两类结构的详细介绍。

1. 分支结构

分支结构使得程序根据不同的条件选择不同的执行路径。C++ 中常用的分支结构有 `if`、`if-else`、`else-if` 和 `switch` 等。

(1) `if` 语句

`if` 语句根据条件是否为真来执行特定的代码块。

```
if (condition) {
    // 当条件为真时执行的代码
}
```

- 如果 `condition` 为真（非零），则执行大括号中的代码块。
- 如果为假（零），则跳过该代码块。

(2) `if-else` 语句

`if-else` 语句可以根据条件是否为真来选择执行不同的代码块。

```
if (condition) {
    // 当条件为真时执行的代码
} else {
    // 当条件为假时执行的代码
}
```

- 如果 `condition` 为真，则执行 `if` 中的代码块；否则，执行 `else` 中的代码块。

(3) `else-if` 语句

`else-if` 语句用于在多个条件中选择一个执行路径。

```
if (condition1) {
    // 当条件1为真时执行的代码
} else if (condition2) {
    // 当条件2为真时执行的代码
} else {
    // 当条件1和条件2都为假时执行的代码
}
```

- 如果 `condition1` 为真，执行第一个代码块；如果为假，检查 `condition2`。
- 如果 `condition2` 为真，执行相应的代码块；否则执行 `else` 中的代码。

(4) `switch` 语句

`switch` 语句用于多分支选择，当条件值是某个常量（整数、字符或枚举类型）时特别有用。

```
switch (expression) {
    case value1:
        // 当 expression == value1 时执行的代码
        break;
    case value2:
        // 当 expression == value2 时执行的代码
        break;
    default:
        // 当没有匹配的 case 时执行的代码
}
```

- `switch` 根据 `expression` 的值跳转到对应的 `case`，并执行该 `case` 后的代码。
- 如果没有匹配的 `case`，则执行 `default` 后的代码。
- `break` 语句用来退出 `switch` 语句块，避免执行后续的 `case`。

2. 循环结构

循环结构允许程序重复执行某段代码，直到满足特定条件。C++ 中常用的循环结构有 `for` 循环、`while` 循环和 `do-while` 循环。

(1) `for` 循环

`for` 循环用于已知次数的循环。

```
for (initialization; condition; increment) {
    // 每次迭代执行的代码
}
```

- `initialization`：初始化循环变量，通常在循环开始时执行一次。
- `condition`：每次循环开始前检查该条件，条件为真时继续执行，条件为假时结束循环。
- `increment`：每次循环结束后执行的代码，通常用于更新循环变量。

例如：

```
for (int i = 0; i < 10; i++) {
    std::cout << i << std::endl;
}
```

该循环将输出从 0 到 9 的数字。

(2) `while` 循环

`while` 循环用于条件判断在循环开始前执行的情况。

```
while (condition) {
    // 当 condition 为真时执行的代码
}
```

- 循环开始前检查 `condition`，如果为真，则执行代码块。否则，循环结束。

例如：

```cpp
int i = 0;
while (i < 10) {
    std::cout << i << std::endl;
    i++;
}
```

该循环同样输出从 0 到 9 的数字。

(3) `do-while` 循环

`do-while` 循环与 `while` 循环类似，但它保证代码至少执行一次，因为条件判断是在循环体之后进行的。

```cpp
do {
    // 循环执行的代码
} while (condition);
```

- 首先执行一次代码块，然后检查 `condition` 是否为真，若为真，则继续执行循环。

例如：

```cpp
int i = 0;
do {
    std::cout << i << std::endl;
    i++;
} while (i < 10);
```

3. 跳转语句

在分支和循环结构中，跳转语句用来控制程序的执行流。常用的跳转语句有 `break`、`continue` 和 `goto`。

(1) `break` 语句

`break` 用于立即退出当前的循环或 `switch` 语句。

```cpp
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;  // 跳出循环
    }
    std::cout << i << std::endl;
}
```

该循环会输出 0 到 4 的数字，因为当 `i` 等于 5 时，`break` 语句跳出循环。

(2) `continue` 语句

`continue` 用于跳过当前的循环迭代，立即进行下一次迭代。

```cpp
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        continue;  // 跳过当前迭代
    }
    std::cout << i << std::endl;
}
```

该循环会输出 0 到 9 的数字，但跳过了 5。

(3) `goto` 语句

`goto` 用于无条件跳转到程序中的其他位置，通常不推荐使用，因为它会让代码变得难以理解和维护。

```cpp
goto label;

label:
// 执行代码
```

总结

- **分支结构**：用来根据条件决定执行路径，主要有 `if`、`if-else`、`else-if` 和 `switch`。
- **循环结构**：用来重复执行代码，直到条件不再满足，主要有 `for`、`while` 和 `do-while`。
- **跳转语句**：用来控制程序流程的跳转，主要有 `break`、`continue` 和 `goto`。

这些控制结构是 C++ 中编写程序时常用的基础工具，掌握它们对理解程序逻辑和控制流程至关重要。

示例：LED 灯控制程序

假设我们希望通过按下一个按钮，来控制一个 LED 灯的开关。具体逻辑是：

- 如果按钮按下（按钮接地），LED 灯就点亮；否则，LED 灯熄灭。
- 同时，我们希望每 500ms 切换一次 LED 灯的状态（如果按钮没有按下）。

程序逻辑：

1. 按钮按下时，LED 亮。
2. 按钮没有按下时，每隔 500ms 切换一次 LED 状态（闪烁 LED）。

C++ 代码：

```cpp
#include <iostream>

using namespace std;

// 模拟按钮和LED的状态（0: 按钮按下/LED亮, 1: 按钮未按下/LED灭）
bool BUTTON = true;  // 初始状态，假设按钮没有按下
bool LED = false;    // 初始状态，假设 LED 关闭

int main() {
```

```
    while (true) {
        // 判断按钮是否按下
        if (BUTTON == false) {  // 按钮按下，LED 点亮
            LED = true;
            cout << "LED is ON\n";
        } else {  // 按钮未按下，LED 闪烁
            // 每 500ms 切换一次 LED 状态
            LED = !LED;
            cout << "LED is " << (LED ? "ON" : "OFF") << "\n";
            delay_ms(500);  // 延时 500ms
        }
    }
    return 0;
}
```