

B.Sc. Computer Science Unit 2

C++ में **Conditional Statements** का उपयोग किसी शर्त के आधार पर किसी विशेष कोड को चलाने के लिए किया जाता है। इनका उपयोग तब किया जाता है जब हमें किसी शर्त को सत्य (True) या असत्य (False) के आधार पर निर्णय लेना हो। C++ में मुख्य रूप से 3 प्रकार के Conditional Statements होते हैं:

1. if Statement

`if` statement का उपयोग तब किया जाता है जब हमें एक शर्त के आधार पर कुछ क्रियाएँ (actions) करना होती हैं। अगर शर्त सत्य होती है, तो कोड का वह हिस्सा चलेगा। अगर शर्त असत्य होती है, तो वह हिस्सा स्किप कर दिया जाएगा।

Syntax:

```
if (condition) {  
    // Code to be executed if condition is true  
}
```

Example:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int a = 10;  
    if (a > 5) {  
        cout << "a is greater than 5" << endl;  
    }  
    return 0;  
}
```

यहां पर अगर `a` की value 5 से बड़ी है, तो "a is greater than 5" print होगा।

2. if-else Statement

`if-else` statement का उपयोग तब किया जाता है जब हमें दो विकल्पों में से एक को चुनना हो। यदि शर्त सत्य है तो `if` के अंदर का कोड चलेगा, और यदि शर्त असत्य है तो `else` के अंदर का कोड चलेगा।

Syntax:

```
if (condition) {  
    // Code to be executed if condition is true  
} else {  
    // Code to be executed if condition is false  
}
```

Example:

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

int main() {
    int a = 3;
    if (a > 5)
    {
        cout << "a is greater than 5" << endl;
    }
else
{
    cout << "a is not greater than 5" << endl;
}
return 0;
}
```

यहां पर, चूंकि a की value 5 से कम है, "a is not greater than 5" print होगा।

3. else-if Statement

else-if का उपयोग तब किया जाता है जब एक से अधिक शर्तें होती हैं और हमें इनमें से एक शर्त पर निर्णय लेना होता है। यह if-else के साथ मिलकर काम करता है, और आप एक से अधिक शर्तों की जांच कर सकते हैं।

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 above conditions are true
}
```

Example:

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

int main() {
    int a = 7;
    if (a > 10) {
        cout << "a is greater than 10" << endl;
    } else if (a == 7) {
        cout << "a is equal to 7" << endl;
    } else {
        cout << "a is less than 7" << endl;
    }
    return 0;
}
```

यहां पर, चूंकि a की value 7 है, "a is equal to 7" print होगा।

4. Nested if Statement

Nested if का मतलब होता है कि आप एक if statement को दूसरे if statement के अंदर रख सकते हैं। इसका उपयोग तब किया जाता है जब एक शर्त के अंदर दूसरी शर्त की जांच करनी होती है।

Syntax:

```
if (condition1) {  
    if (condition2) {  
        // Code to be executed if condition2 is true  
    }  
}
```

Example:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int a = 10;  
    int b = 5;  
    if (a > 5) {  
        if (b < 10) {  
            cout << "a is greater than 5 and b is less than 10" << endl;  
        }  
    }  
    return 0;  
}
```

यहां पर, शर्त a > 5 सत्य होने पर ही दूसरे if statement की शर्त b < 10 की जांच होगी।

5. Switch Statement

switch statement का उपयोग तब किया जाता है जब हमें एक ही variable के कई संभावित मान (values) की जांच करनी होती है। यह खासकर तब उपयोगी होता है जब if-else के साथ कई शर्तों की तुलना करनी हो, जिससे कोड थोड़ा जटिल हो सकता है।

Syntax:

```
switch (variable) {  
    case value1:  
        // Code to be executed if variable == value1  
        break;  
    case value2:  
        // Code to be executed if variable == value2  
        break;  
    default:  
        // Code to be executed if no case matches
```

```
}
```

Example:

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

int main() {
    int day = 3;
    switch (day) {
        case 1:
            cout << "Monday" << endl;
            break;
        case 2:
            cout << "Tuesday" << endl;
            break;
        case 3:
            cout << "Wednesday" << endl;
            break;
        default:
            cout << "Invalid day" << endl;
    }
    return 0;
}
```

यहां पर, day की value 3 होने के कारण "Wednesday" print होगा।

Conclusion

C++ में Conditional Statements का उपयोग हमारे कोड को अधिक गतिशील और flexible बनाने के लिए किया जाता है। इनका उपयोग करके हम विभिन्न परिस्थितियों के आधार पर कोड का प्रवाह नियंत्रित कर सकते हैं।

Iterative statements in c++

C++ में **Iterative Statements** का उपयोग तब किया जाता है जब हमें किसी कोड को बार-बार (multiple times) चलाना हो। इसका मतलब है कि हम किसी क्रिया को एक निश्चित संख्या में या तब तक दोहराते हैं जब तक कोई शर्त सत्य (True) न हो। C++ में मुख्य रूप से 3 प्रकार के **Iterative Statements** होते हैं:

1. for Loop

`for loop` का उपयोग तब किया जाता है जब हमें एक निश्चित संख्या में किसी कार्य को दोहराना हो। इसमें तीन मुख्य हिस्से होते हैं:

1. Initialization (शुरूआत में एक वैरिएबल की value सेट करना)
2. Condition (शर्त, जिससे loop चलता रहता है)
3. Increment/Decrement (वैरिएबल की value को बढ़ाना या घटाना)

Syntax:

```
for (initialization; condition; increment/decrement) {  
    // Code to be executed repeatedly  
}
```

Example:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    for (int i = 1; i <= 5; i++) {  
        cout << "i = " << i << endl;  
    }  
    return 0;  
}
```

यहां पर, `i` की value 1 से शुरू होती है और हर बार loop के अंदर बढ़ती है (`i++`)। जब तक `i` की value 5 से कम या बराबर होती है, तब तक loop चलता रहेगा। Output में `i = 1, i = 2, ..., i = 5` दिखेगा।

2. while Loop

`while loop` का उपयोग तब किया जाता है जब हमें किसी शर्त के सत्य होने तक कोई कार्य दोहराना हो। यह तब तक चलता रहता है जब तक दी गई शर्त सत्य रहती है।

Syntax:

```
while (condition) {  
    // Code to be executed repeatedly  
}
```

Example:

```
#include <iostream>
```

```

using namespace std;

int main() {
    int i = 1;
    while (i <= 5) {
        cout << "i = " << i << endl;
        i++; // increment
    }
    return 0;
}

```

यहां पर, `i` की value 1 से शुरू होती है और हर iteration में बढ़ती है। जब तक `i <= 5` शर्त सत्य रहती है, तब तक loop चलता रहेगा। Output में `i = 1, i = 2, ..., i = 5` दिखेगा।

3. do-while Loop

`do-while loop` का उपयोग तब किया जाता है जब हमें किसी कार्य को कम से कम एक बार चलाना हो, फिर शर्त की जांच करनी हो। इसका मतलब है कि कोड का ब्लॉक एक बार तो जरूर चलेगा, फिर शर्त की जांच होगी।

Syntax:

```

do {
    // Code to be executed at least once
} while (condition);

```

Example:

```

#include <iostream>
using namespace std;

int main() {
    int i = 1;
    do {
        cout << "i = " << i << endl;
        i++; // increment
    } while (i <= 5);
    return 0;
}

```

यहां पर भी `i` की value 1 से शुरू होती है। चूंकि यह `do-while loop` है, इसलिए कोड कम से कम एक बार चलेगा और फिर शर्त `i <= 5` की जांच करेगा। Output में `i = 1, i = 2, ..., i = 5` दिखेगा।

4. Nested Loops

Nested Loops का मतलब है कि आप एक loop को दूसरे loop के अंदर रख सकते हैं। इसका उपयोग तब किया जाता है जब हमें 2D या मल्टीडायमेंशनल डेटा के साथ काम करना हो, जैसे एक matrix के साथ काम करते समय।

Syntax:

```

for (initialization1; condition1; increment/decrement1) {
    for (initialization2; condition2; increment/decrement2) {
        // Code to be executed repeatedly for inner loop
    }
}

```

}

Example:

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

int main() {
    for (int i = 1; i <= 3; i++) {
        for (int j = 1; j <= 2; j++) {
            cout << "i = " << i << ", j = " << j << endl;
        }
    }
    return 0;
}
```

यहां पर, बाहरी loop में i की value 1 से 3 तक जाएगी, और अंदर के loop में j की value 1 से 2 तक जाएगी। इसका Output होगा:

```
ini
Copy
i = 1, j = 1
i = 1, j = 2
i = 2, j = 1
i = 2, j = 2
i = 3, j = 1
i = 3, j = 2
```

Conclusion

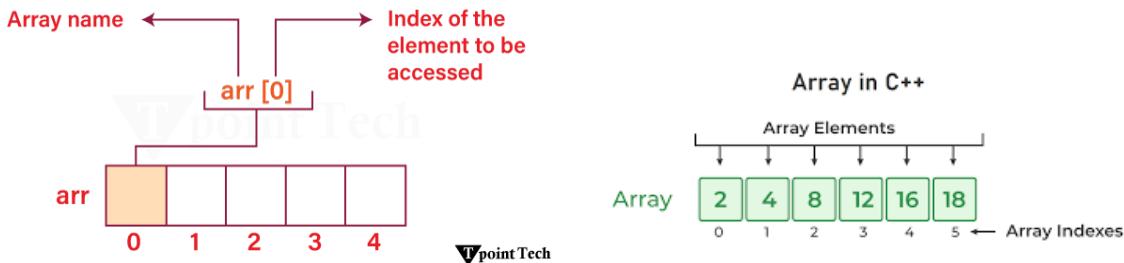
C++ में **Iterative Statements** का उपयोग तब किया जाता है जब हमें किसी कोड को एक या अधिक बार चलाने की आवश्यकता होती है। इनका उपयोग हम तब करते हैं जब हमें डेटा के एक सेट को process करना हो या जब तक कोई शर्त सत्य रहे, तब तक किसी कार्य को दोहराना हो। **for**, **while**, और **do-while loops** सबसे आम iterative statements हैं, और इनके जरिए हम अपने प्रोग्राम को अधिक प्रभावी और लचीला बना सकते हैं।

while और **do-while** में अंतर :

Aspect	while loop	do-while loop
Condition Check	शर्त पहले चेक होती है। (Pre-condition check)	कोड पहले execute होता है, फिर शर्त चेक होती है। (Post-condition check)
Execution Guarantee	यदि शर्त पहली बार असत्य (False) हो तो loop नहीं चलेगा।	कम से कम एक बार कोड execute होगा, भले ही शर्त असत्य हो।
Use Case	जब शर्त पहले से सत्य हो और तब तक काम करना हो। जब तक शर्त पूरी न हो।	जब आपको कम से कम एक बार कोड चलाना हो, फिर शर्त की जांच करनी हो।
Examples	- डेटा पढ़ते समय यदि शर्त सत्य हो। - किसी कंडीशन के आधार पर कोड को बार-बार चलाना।	- Menu-driven programs - जब आपको कोई ऑपरेशन कम से कम एक बार करना हो।

Array

C++ में **Array** एक डेटा संरचना (data structure) है जो समान प्रकार के तत्वों (elements) को एक साथ संग्रहित करने के लिए उपयोग की जाती है। एक array को हम एक सूची (list) के रूप में समझ सकते हैं, जिसमें सभी तत्व समान प्रकार के होते हैं और इनका आकार (size) पहले से निर्धारित होता है।



Array की प्रमुख विशेषताएँ:

- Fixed Size:** Array का आकार स्थिर होता है, यानी कि एक बार array बना लेने के बाद इसका आकार बदल नहीं सकता।
- Indexing:** Array के प्रत्येक तत्व को एक इंडेक्स (index) द्वारा एक्सेस किया जाता है। C++ में array का इंडेक्स 0 से शुरू होता है। उदाहरण के लिए, यदि आपके पास एक array है arr[5], तो आप arr[0], arr[1], ..., arr[4] तक के तत्वों को एक्सेस कर सकते हैं।
- Same Data Type:** Array के सभी तत्व समान प्रकार (data type) के होते हैं। उदाहरण के लिए, अगर आपने integer type array बनाया है तो उस array में केवल integers ही होंगे।
- Memory Allocation:** Array का आकार तय होने पर, कंप्यूटर में एक contiguous memory block में सभी elements संग्रहित होते हैं।

Array का Declaration:

C++ में array को इस प्रकार घोषित किया जाता है:

```
datatype arrayName[size];
```

जहाँ:

- datatype:** वह डेटा प्रकार (data type) है जिसे array स्टोर करेगा, जैसे int, float, char आदि।
- arrayName:** यह array का नाम है।
- size:** array में कितने तत्व होंगे, यह दर्शाता है।

उदाहरण:

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

int main() {
    // Integer type का array जो 5 तत्वों को संग्रहित करेगा
    int arr[5] = {1, 2, 3, 4, 5};
```

```

// Array के तत्वों को प्रदर्शित करना
for (int i = 0; i < 5; i++) {
    cout << arr[i] << " ";
}
return 0;
}

```

Output:

Copy
1 2 3 4 5

Array के प्रकार:

- One-Dimensional Array:** एक-आयामी array (1D array) में तत्वों की एक ही रेखा होती है।
- Two-Dimensional Array:** दो-आयामी array (2D array) में तत्वों की एक तालिका होती है, जैसे rows और columns।
- Multi-Dimensional Array:** इससे भी अधिक आयामों वाले array हो सकते हैं, जैसे 3D array।

Two-Dimensional Array Example:

	column 1	column 2	column 3	column 4	column 5
row1	arr[0][0]	arr[0][1]	arr[0][2]	arr[0][3]	arr[0][4]
row2	arr[1][0]	arr[1][1]	arr[1][2]	arr[1][3]	arr[1][4]
row3	arr[2][0]	arr[2][1]	arr[2][2]	arr[2][3]	arr[2][4]

```

#include <iostream>
using namespace std;

int main() {
    // 2D array में 3 rows और 4 columns होते हैं
    int arr[3][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };

    // 2D array के तत्वों को प्रदर्शित करना
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}

```

Output:

```
1 2 3 4  
5 6 7 8  
9 10 11 12
```

Array का उपयोग:

- Array का उपयोग तब किया जाता है जब हमें एक ही प्रकार के बहुत सारे डेटा को संग्रहित और प्रोसेस (process) करना हो।
- यह सामान्यतः mathematical calculations, storing student marks, storing elements in a list, etc. के लिए उपयोग किया जाता है।

Advantages of Arrays:

1. **Fast Access:** Array में किसी भी तत्व तक पहुँचने में समय बहुत कम लगता है क्योंकि यह contiguous memory block में होते हैं।
2. **Memory Efficiency:** Array में समान प्रकार के डेटा को एक ही स्थान पर संग्रहित किया जाता है, जिससे memory का efficient उपयोग होता है।

Disadvantages of Arrays:

1. **Fixed Size:** Array का आकार तय होता है, और इसे runtime पर बदलना संभव नहीं होता।
2. **Inefficient for Insertion/Deletion:** Array में तत्वों को जोड़ना या हटाना थोड़ा मुश्किल होता है, खासकर जब array भर चुका हो।

(Definition of Functions in C++)

C++ में फ़ंक्शन कोड का एक ब्लॉक होता है, जिसे एक विशेष नाम दिया गया होता है। यह ब्लॉक किसी विशेष कार्य को निष्पादित करने के लिए बनाया जाता है। जब भी यह कार्य करना होता है, तो उस फ़ंक्शन को उसके नाम से कॉल किया जाता है।

उदाहरण: **main()** फ़ंक्शन हर C++ प्रोग्राम का मुख्य फ़ंक्शन होता है।

फ़ंक्शन के लाभ (Advantages of Functions)

1. **कोड का पुनः उपयोग (Code Reusability):** एक बार बनाए गए फ़ंक्शन को बार-बार उपयोग किया जा सकता है।
2. **कोड को सरल बनाना (Simplifies Code):** लंबे कोड को छोटे-छोटे हिस्सों में विभाजित करता है।
3. **समस्या का विभाजन (Modularity):** बड़े प्रोग्राम को छोटे मॉड्यूल्स में विभाजित करता है।
4. **डिबगिंग आसान बनाता है (Ease of Debugging):** यदि कोई समस्या हो, तो उसे फ़ंक्शन के स्तर पर खोजा जा सकता है।
5. **समझने में सरल (Easy to Understand):** प्रोग्राम को पढ़ना और समझना आसान हो जाता है।
6. **अत्यधिक लचीलापन (Flexibility):** पैरामीटर्स और रिटर्न वैल्यू के माध्यम से अन्य फ़ंक्शन्स से डेटा आदान-प्रदान कर सकते हैं।

फ़ंक्शन का सिटेक्स (Syntax):

```
return_type function_name(parameters)
{
    // Function body
    return value; // यह वैकल्पिक है (Optional)
}
```

फंक्शन के हिस्से:

1. **Return Type** (वापसी का प्रकार):
 - फंक्शन किस प्रकार का डेटा लौटाएगा, यह निर्धारित करता है।
 - जैसे: **int**, **float**, **void** (अगर कोई डेटा नहीं लौटाना है)।
2. **Function Name** (फंक्शन का नाम):
 - फंक्शन का नाम जो उसकी पहचान करता है। इसे अर्थपूर्ण और स्पष्ट रखें।
3. **Parameters** (पैरामीटर):
 - इनपुट मान जो फंक्शन को पास किए जाते हैं। यह वैकल्पिक हो सकते हैं।
4. **Function Body** (फंक्शन का शरीर):
 - वह कोड जो फंक्शन के अंदर लिखा जाता है और कार्य पूरा करता है।
5. **Return Statement** (वापसी कथन):
 - फंक्शन से परिणाम वापस भेजने के लिए उपयोग किया जाता है। यह **void** फंक्शन के लिए वैकल्पिक है।

फंक्शन के प्रकार (**Types of Functions in C++**)

1. **लाइब्रेरी फंक्शन (Library Functions)/built-in/ pre define):**
 - प्रोग्रामिंग के लिए पहले से परिभाषित फंक्शन्स।
 - उदाहरण: **sqrt()**, **pow()**, **strlen()**, आदि।
 - ये **<cmath>**, **<cstring>** जैसी हेडर फाइल्स में परिभाषित होते हैं।
2. **यूजर-डिफाइंड फंक्शन (User-Defined Functions):**
 - उपयोगकर्ता द्वारा बनाए गए फंक्शन्स।

उदाहरण:

```
void greet() {  
    cout << "Hello, Students!";  
}  
  
यूजर-डिफाइंड फंक्शन का उदाहरण:  
#include <iostream>  
using namespace std;  
  
// दो संख्याओं का योग करने का फंक्शन  
int add(int num1, int num2) {  
    return num1 + num2;  
}  
  
int main() {  
    int a = 5, b = 10;  
    cout << "योग: " << add(a, b) << endl;  
    return 0;  
}
```

Function Prototypes (फंक्शन प्रोटोटाइप):

- फंक्शन के उपयोग से पहले उसकी घोषणा।
- यह कंपाइलर को फंक्शन के बारे में जानकारी देता है।

उदाहरण:

```
int add(int, int);
```

फंक्शन कॉल करना (**Calling a Function**):

फंक्शन को उसके नाम और आवश्यक पैरामीटर के साथ बुलाया जाता है।

उदाहरण:

```
cout << add(3, 7);
```

पैरामीटर पास करने के प्रकार:

बिंदु	विवरण
1. कॉल बाय वैल्यू (Call by Value)	<p>वास्तविक पैरामीटर की एक प्रति पास की जाती है। फ़ंक्शन के अंदर किया गया परिवर्तन मूल मान को प्रभावित नहीं करता।</p> <p>उदाहरण:</p> <pre>void modify(int x) { x = 10; }</pre>
2. कॉल बाय रेफरेंस (Call by Reference)	<p>वास्तविक पैरामीटर को रेफरेंस के रूप में पास किया जाता है। फ़ंक्शन के अंदर किया गया परिवर्तन मूल मान को प्रभावित करता है।</p> <p>उदाहरण:</p> <pre>void modify(int &x) { x = 10; }</pre>
1. बिना पैरामीटर और रिटर्न टाइप के फ़ंक्शन	<p>उदाहरण:</p> <pre>void greet() { cout << "Hello, World!"; }</pre>
3. पैरामीटर और रिटर्न टाइप के साथ	<p>उदाहरण:</p> <pre>int square(int num) { return num * num; }</pre>

Recursive Functions (पुनरावृत्त फ़ंक्शन):

- ऐसा फ़ंक्शन जो खुद को कॉल करता है।

उदाहरण:

```
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

फ़ंक्शन के फायदे:

- कोड पुनः उपयोग: एक बार लिखें और कई बार उपयोग करें।
- डीबग करना आसान: छोटे-छोटे भागों में त्रुटि खोजना सरल है।
- मॉड्यूलर प्रोग्रामिंग: बड़े प्रोग्राम को संभालने में मदद।
- रीयूजेबिलिटी: फ़ंक्शन दूसरे प्रोग्राम में भी उपयोग किए जा सकते हैं।

उदाहरण: फ़ैक्टोरियल निकालने का प्रोग्राम

```
#include <iostream>
using namespace std;
int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

```
int main() {
```

```
int num;  
cout << "संख्या दर्ज करें: ";  
cin >> num;  
cout << "फैक्टोरियल: " << factorial(num) << endl;  
return 0;  
}
```

आउटपुट:

अगर इनपुट 5 है, तो आउटपुट होगा:

फैक्टोरियल: 120

3. Array as Function Argument (Array को फ़ंक्शन में पास करना)

परिभाषा:

जब हम कोई **array** किसी फ़ंक्शन को पास करते हैं, तो वो हमेशा **reference** के रूप में ही पास होता है। मतलब, अगर हम फ़ंक्शन में **array** को बदलते हैं, तो वो असली **array** में भी बदल जाता है।

उदाहरण:

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

// Array दिखाने वाला फ़ंक्शन
void display(int arr[], int size) {
    cout << "Array की वैल्यू:";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

// Array को अपडेट करने वाला फ़ंक्शन
void updateArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i] += 5;
    }
}

int main() {
    int nums[3] = {10, 20, 30};

    cout << "Original Array:" << endl;
    display(nums, 3);

    updateArray(nums, 3); // Array पास किया गया

    cout << "Updated Array:" << endl;
    display(nums, 3);

    return 0;
}
```

विश्लेषण:

- **arr[]** को जब पास किया जाता है, तो उसका **address** फ़ंक्शन को मिलता है।
- इसलिए **arr[i] += 5;** से असली **array** बदल जाता है।

Output:

Original Array:

Array की वैल्यू: 10 20 30

Updated Array:

Array की वैल्यू: 15 25 35