ICPC
MNU COMMUNINTY

LOOPS

PREPERED BY: YOUSSEF BAHAA

Agenda

- INTRODUCTION
- WHAT ARE THE LOOPS
- TYPES OF LOOPS
- FOR LOOP
- WHILE LOOP
- DO WHILE LOOP
- PRACTICE

LOOPS

WHAT ARE LOOPS?

- A LOOP IS A WAY TO REPEAT A BLOCK OF CODE MULTIPLE TIMES WITHOUT WRITING IT OVER AND OVER.
- INSTEAD OF COPYING THE SAME CODE 10 TIMES, YOU CAN WRITE IT ONCE AND LET THE LOOP RUN IT.

LOOPS

WHY DO WE NEED LOOPS?

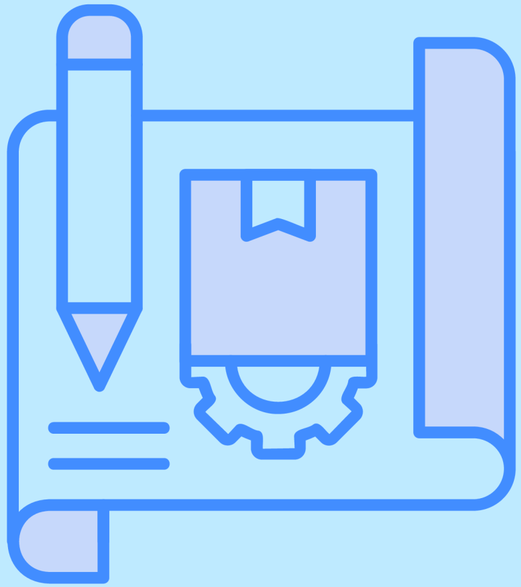
- TO SAVE TIME AND REDUCE REPETITION IN CODE.
- TO HANDLE SITUATIONS WHERE WE DON'T KNOW HOW MANY TIMES AN ACTION WILL REPEAT (E.G., KEEP ASKING THE USER FOR INPUT UNTIL THEY TYPE EXIT).
- TO WORK WITH COLLECTIONS LIKE ARRAYS, STRINGS, OR VECTORS (E.G., PRINT ALL CHARACTERS IN A WORD).

HOW LOOPS WORKS

- A LOOP USUALLY HAS 3 PARTS:

1. INITIALIZATION → WHERE WE START.
2. CONDITION → WHEN TO STOP.
3. UPDATE → HOW TO MOVE FORWARD.

- WHILE THE CONDITION IS TRUE → THE LOOP RUNS.
- WHEN THE CONDITION BECOMES FALSE → THE LOOP STOPS.
- A SIMPLE EXAMPLE WITHOUT A LOOP



EXAMPLE

A SIMPLE EXAMPLE WITHOUT A LOOP

main.cpp

Share Run

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

int main() {
    cout << 1 << " ";
    cout << 2 << " ";
    cout << 3 << " ";
    cout << 4 << " ";
    cout << 5 << " ";
}

```

Output

```
1 2 3 4 5
```

INTRO TO LOOPS

USING LOOP

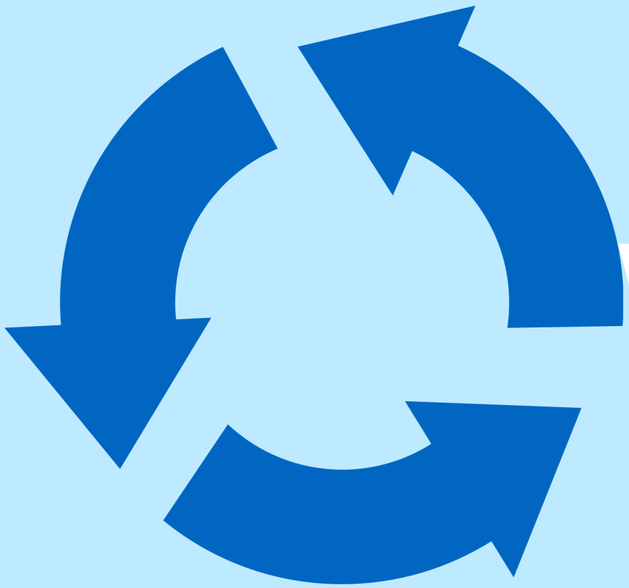
main.cpp

Share Run

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

int main() {
    for (int i = 1; i <= 5; i++) {
    cout << i << " ";
    }
}

```

Output

```
1 2 3 4 5
```



WHILE LOOPS

WHILE

THE WHILE LOOP

THE WHILE LOOP IN C++ MAINLY COMES IN TWO FORMS:

A) WHILE LOOP

B) DO-WHILE LOOP

LOGIC:

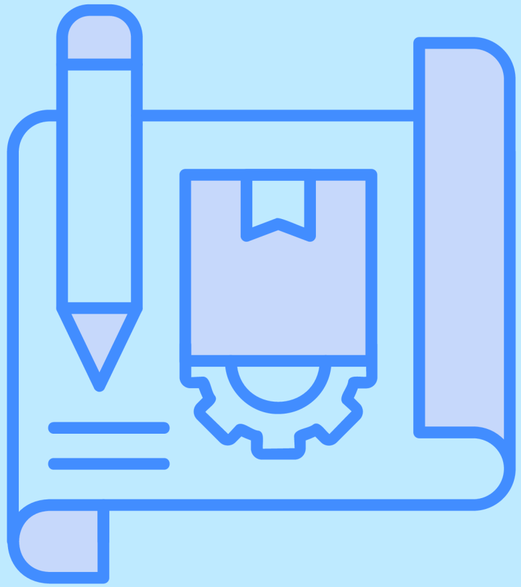
- CHECKS THE CONDITION FIRST.
- IF TRUE → RUNS THE BLOCK.
- IF FALSE → IT STOPS IMMEDIATELY.

SYTNAX

main.cpp

Share Run

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

int main() {
    while (condition) {
    // code to run
    }
}

```



EXAMPLE

main.cpp

Share Run

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

int main() {
    int i = 1;
    while (i <= 5) {
    cout << i << " ";
    i++;
    }
}

```

Output

1 2 3 4 5

TIPS

- DON'T FORGET TO UPDATE YOUR VARIABLE (I++) → OTHERWISE INFINITE LOOP.
- THE WHILE LOOP IS BEST USED WHEN YOU DON'T KNOW EXACTLY HOW MANY TIMES TO REPEAT (E.G., READING INPUT UNTIL USER QUITS).

DO-WHILE

LOGIC

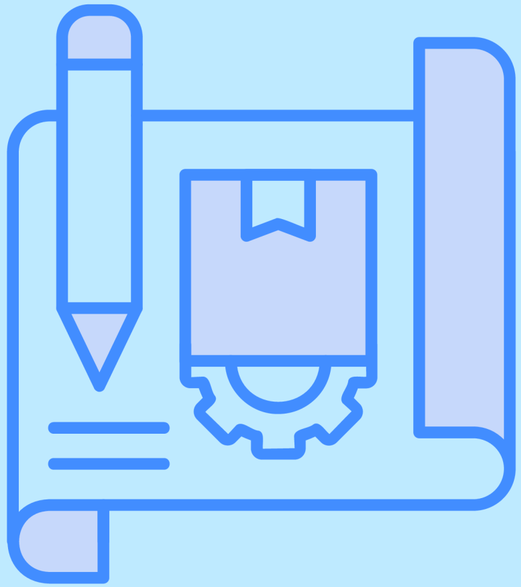
- RUNS THE BLOCK AT LEAST ONCE, THEN CHECKS THE CONDITION.
- KEEPS REPEATING AS LONG AS CONDITION IS TRUE.

SYTNAX

main.cpp

Share Run

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

int main() {
    do {
    // code to run
    }
    while (condition);
}

```



EXAMPLE

main.cpp

Share Run

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

int main() {
    int i = 1;
do {
    cout << i << " ";
    i++;
    }
while (i <= 5);
}
```

Output

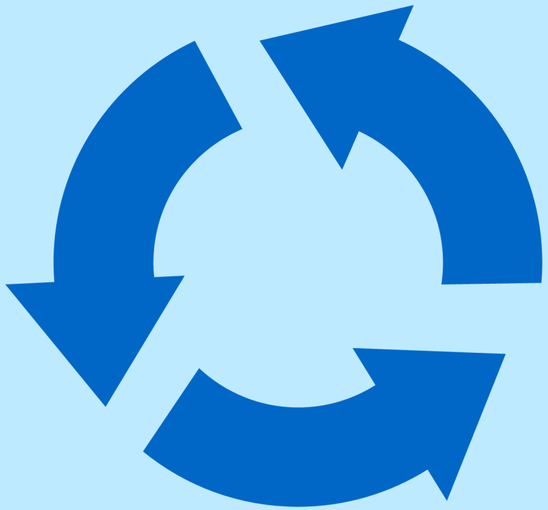
```
1 2 3 4 5
```

TIPS

- EVEN IF CONDITION IS FALSE FROM THE START, THE LOOP EXECUTES ONCE.
- USEFUL WHEN YOU NEED TO SHOW SOMETHING FIRST, THEN DECIDE WHETHER TO CONTINUE (E.G., MENU OPTIONS).

KEY DIFFERENCE (WHILE VS DO-WHILE)

Feature	While Loop	Do-While Loop
Condition check timing	Before loop body	After loop body
Minimum runs	0 times (may never run)	1 time (always runs once)
Common use	When you want to check first	When you must run at least once



FOR LOOP

FOR LOOP

THE FOR LOOP IN C++ MAINLY ALSO COMES IN TWO FORMS:

A) INDEXED-BASED FOR LOOP

- BEST WHEN YOU KNOW EXACTLY HOW MANY TIMES TO LOOP.
- USES INITIALIZATION → CONDITION → UPDATE.

B) RANGE-BASED FOR LOOP

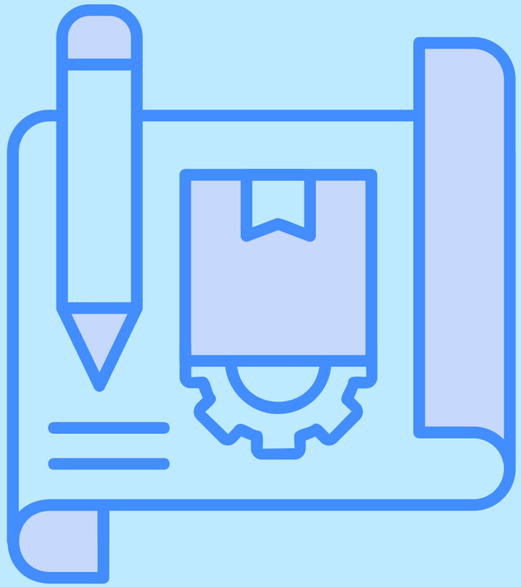
- LOOPS DIRECTLY OVER ELEMENTS IN A CONTAINER (STRING, ARRAY, VECTOR, ETC.).
- CLEANER AND LESS ERROR-PRONE.

SYTNAX (INDEXED-BASED)

main.cpp

Share Run

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

int main() {
    for (initialization; condition; update) {
    // code
    }
}
```



EXAMPLE

main.cpp

Share Run

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

int main() {
    for (int i = 1; i <= 5; i++) {
    cout << i << " ";
    }
}
```

Output

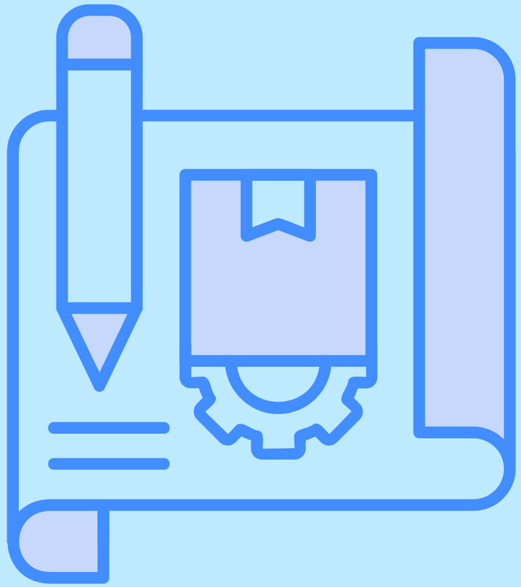
```
1 2 3 4 5
```

SYTNAX (RANGE-BASED)

main.cpp

Share Run

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

int main() {
    for (datatype variable : container) {
    // code
    }
}

```



EXAMPLE

main.cpp

Share Run

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

int main() {
    string word = "Omar";
    for (char c : word) {
    cout << c << " ";
    }
}

```

Output

```
O m a r
```

TIP: USE AUTO WHEN THE TYPE IS UNKNOWN OR LONG

main.cpp

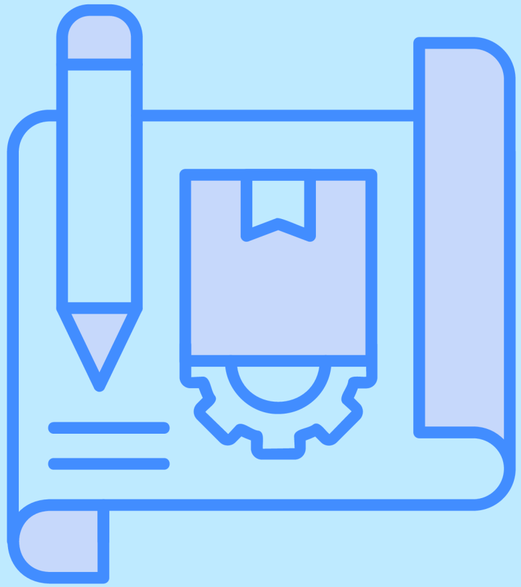
Share Run

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

int main() {
    for (auto x : myVector) {
    cout << x << " ";
    }
}

```

NESTED FOR LOOPS

LOGIC

- A LOOP INSIDE ANOTHER LOOP.
- THE INNER LOOP RUNS FULLY FOR EACH STEP OF THE OUTER LOOP.
- USEFUL FOR TABLES, GRIDS, 2D ARRAYS, AND PATTERNS.



EXAMPLE

INDEX-BASED FOR

main.cpp

Share Run

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

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

Output

```
(1,1) (1,2) (2,1) (2,2) (3,1) (3,2)
```

RANGE-BASED FOR

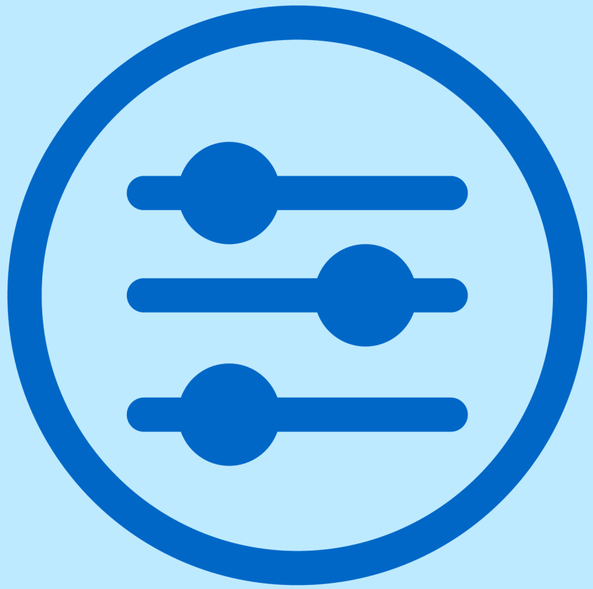
main.cpp

Share Run

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

int main() {
    string word = "Hi";
for (char c1 : word) {
    for (char c2 : word) {
        cout << c1 << c2 << " ";
        }
    }
}
```

Output

```
HH Hi iH ii
```



WAYS TO CONTROL LOOPS

- WHEN WRITING LOOPS, SOMETIMES WE DON'T WANT THEM TO RUN STRICTLY FROM START TO FINISH.
- WE MAY NEED TO STOP EARLY, SKIP CERTAIN CASES, OR JUMP OUT OF NESTED LOOPS.

C++ PROVIDES TWO CONTROL STATEMENTS FOR THIS:

A) **BREAK**

PURPOSE:

- IMMEDIATELY STOPS THE LOOP (NO MORE ITERATIONS).
- CONTROL JUMPS TO THE FIRST STATEMENT AFTER THE LOOP.

WORKS IN:

- FOR, WHILE, DO-WHILE, AND EVEN SWITCH.

B) **CONTINUE**

PURPOSE:

- SKIPS THE REST OF THE CURRENT ITERATION.
- GOES BACK TO THE NEXT CYCLE OF THE LOOP.

DIFFERENCE BY LOOP TYPE:

- FOR LOOP: AFTER CONTINUE, IT STILL PERFORMS THE UPDATE STEP (I++), THEN CHECKS CONDITION.
- WHILE / DO-WHILE: AFTER CONTINUE, IT JUMPS DIRECTLY TO CONDITION CHECK (NO UPDATE STEP HAPPENS AUTOMATICALLY).

BREAK

WHILE EXAMPLE

main.cpp

Share Run

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

int main() {
    int x = 0;
while (x < 10) {
    if (x == 5) break;  // stop early
    cout << x << " ";
    x++;
    }
}

```

Output

```
0 1 2 3 4
```

DO-WHILE EXAMPLE

main.cpp

Share Run

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

int main() {
    int x = 0;
do {
    if (x == 0) break;  // breaks immediately
    cout << "Hello";
    } while (x < 5);
}

```

NOTHING BECAUSE THE X = 0

Output

FOR EXAMPLE

main.cpp

Share Run

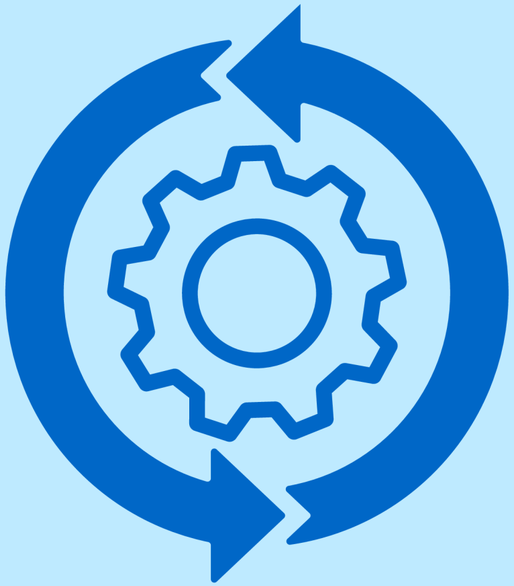
```cpp
#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 10; i++) {
    if (i == 4) break;
    cout << i << " ";
    }
}

```

Output

```
1 2 3
```

REAL-LIFE USE CASE:

SEARCHING FOR AN ITEM IN A LIST → ONCE FOUND, NO NEED TO KEEP LOOPING.



CONTINUE

WHILE EXAMPLE

main.cpp

Share Run

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

int main() {
    int x = 0;
while (x < 5) {
    x++;
    if (x == 3) continue;  // skip 3
    cout << x << " ";
    }
}
```

Output

```
1 2 4 5
```

DO-WHILE EXAMPLE

main.cpp

Share Run

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

int main() {
    int x = 0;
do {
    x++;
    if (x == 2) continue;  // skip 2
    cout << x << " ";
    } while (x < 4);
}

```

Output

```
1 3 4
```

FOR EXAMPLE

main.cpp

Share Run

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

int main() {
    for (int i = 1; i <= 5; i++) {
    if (i % 2 == 0) continue;  // skip even numbers
    cout << i << " ";
    }
}
```

Output

```
1 3 5
```

REAL-LIFE USE CASE:

PROCESSING A LIST BUT SKIPPING INVALID ENTRIES (E.G., NEGATIVE NUMBERS, EMPTY STRINGS).

BREAK & CONTINUE IN NESTED LOOPS

BY DEFAULT, BREAK AND CONTINUE AFFECT ONLY THE INNERMOST LOOP THEY ARE INSIDE.

main.cpp

Share Run

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

int main() {
    for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (j == 2) break;   // breaks only inner loop
        cout << "(" << i << "," << j << ") ";
        }
    }
}
```

Output

```
(1,1) (2,1) (3,1)
```

SUMMARY OF LOOP CONTROL:

- BREAK = STOP THE LOOP COMPLETELY.
- CONTINUE = SKIP THIS TURN, MOVE TO THE NEXT.
- IN NESTED LOOPS, THEY ONLY AFFECT THE CURRENT INNER LOOP.

CODEFORCE TIME !

THANK YOU!

FOLLOW US !