

Conditional Execution and Loops in C

Outline

- 1 Conditional Execution
- 2 Loop
- 3 break and continue
- 4 Nested Loops
- 5 Nested if Statements
- 6 The switch Statement
- 7 Exercise

Conditional Execution

Conditional Execution in C

- **Conditional execution** allows a program to take different actions based on certain conditions
- Conditions are expressed using `if`, `else`, and `else if` statements
- Condition expressions must evaluate to `true` (non-zero) or `false` (zero)

if Statement

Syntax:

```
if(condition){  
    // statements  
}
```

Example:

```
int x = 10;  
if (x > 0) {  
    printf("Positive number\n");  
}
```

if-else Statement

```
if(condition){  
    // commands to execute if true  
} else{  
    // commands to execute if false  
}
```

if-else Example

```
int age = 18;

if(age >= 18){
    printf("Eligible to vote\n");
} else{
    printf("Not eligible to vote\n");
}
```

else if Ladder

```
if(condition1){  
    ...  
} else if(condition2) {  
    ...  
} else{  
    ...  
}
```


else if Ladder Example

```
int marks = 75;

if(marks >= 90){
    printf("Grade A\n");
} else if(marks >= 75){
    printf("Grade B\n");
} else{
    printf("Grade C\n");
}
```

Loop

Loops in C

- Loops are used to execute a block of code repeatedly.
- Types of loops in C:
 - `for` loop: when number of iterations is known
 - `while` loop: when condition is checked before each iteration
 - `do-while` loop: condition checked *after* executing loop body

In `do-while` loop, the body of the loop is always executed at least once.

for Loop

```
for(initialization; condition; update){  
    // statements  
}
```

The elements (initialization, condition and update) inside the for keyword, can be omitted. For example,

- Initialization can be performed before the for keyword
- Condition and update can be moved inside the loop body
- `for(;;){...}` creates an infinite loop

for Loop Example

```
for(int i = 1; i <= 5; i++){  
    printf("%d ", i);  
}
```

for Loop Example (cont.)

```
int i = 1;

for(;;){
    if(i > 5){
        break;
    }
    printf("%d", i);
    i++;
}
```

while Loop

Syntax:

```
while (condition) {  
    // statements  
}
```

Example:

```
int i = 1;  
while(i <= 5){  
    printf("%d ", i);  
    i++;  
}
```

do-while Loop

Syntax:

```
do{  
    // statements  
} while(condition);    // don't forget this semicolon
```

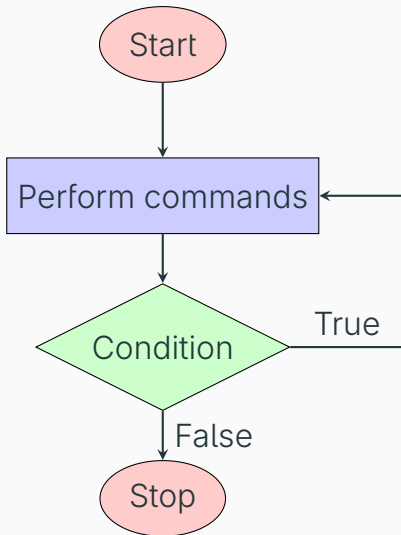
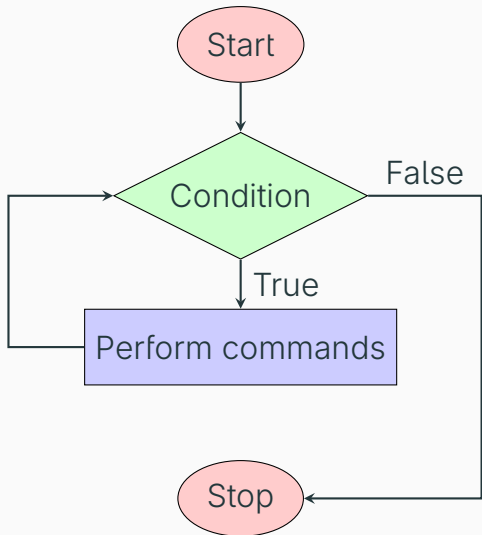
Example:

```
int i = 1;  
do{  
    printf("%d ", i);  
    i++;  
} while(i <= 5);
```


while vs do-while Loop

- `while`: condition checked *before* loop body
- `do-while`: condition checked *after* running the first iteration of the loop, so the loop runs at least once

Flowchart: While vs Do-While



break and continue

The break Statement

- The break statement immediately terminates the loop or switch statement in which it is encountered
- Control of the program then transfers to the statement immediately following the loop or switch
- It is commonly used to exit a loop prematurely based on a certain condition

break Example

```
for(int i = 1; i <= 10; i++){  
    if(i == 5){  
        break;    // Exit the loop when i is 5  
    }  
    printf("%d ", i);  
}  
printf("\nLoop terminated.\n");
```

The continue Statement

- The continue statement skips the remaining statements in the current iteration of a loop and proceeds to the next iteration
- It is used when you want to bypass certain parts of the loop's body for specific conditions without exiting the entire loop

continue Example

```
for(int i = 1; i <= 5; i++){  
    if (i == 3){  
        continue;    // Skip printing when i is 3  
    }  
    printf("%d ", i);  
}  
printf("\nLoop finished.\n");
```

Nested Loops

Nested `if` Statements

The `switch` Statement

Exercise

Exercise

- Write a C program to check whether a number is positive or negative
- Write a C program to check whether an integer is even or odd
- Write a program to print the first n natural numbers using a `for` loop
- Write a program to compute the sum of numbers from 1 to n using a `while` loop
- Write a program to repeatedly take user input (integer) and print its square, until a negative number is entered