5. Flow Control

5.1 Conditional Statements

5.1.1 if Statement

Python vs C Syntax:

Python:

```
if condition:
    statement1
    statement2
elif another_condition:
    statement3
else:
    statement4
```

C:

```
if (condition) {
    statement1;
    statement2;
} else if (another_condition) {
    statement3;
} else {
    statement4;
}
```

Key Differences:

- C requires parentheses around conditions
- C uses curly braces `{}` instead of indentation
- C requires semicolons after statements

5.1.2 Relational Operators

Operator	Meaning	Python	C
`==`	Equal to	`a == b`	`a == b`

Operator	Meaning	Python	C
`!=`	Not equal to	`a != b`	`a != b`
`<`	Less than	`a < b`	`a < b`
`>`	Greater than	`a > b`	`a > b`
`<=`	Less than or equal	`a <= b`	`a <= b`
`>=`	Greater than or equal	`a >= b`	`a >= b`

5.1.3 Logical Operators

Operator	Meaning	Python	C
`&&`	Logical AND	`and` or `&`	`&&`
`||`	Logical OR	`or` or `|`	`||`
`!`	Logical NOT	`not` or `~`	`!`

Examples in C :

```
// Python: if age >= 18 and score > 80:
if (age >= 18 && score > 80) {
    printf("Eligible for scholarship\n");
}

// Python: if not (x < 0 or x > 100):
if (!(x < 0 || x > 100)) {
    printf("Valid percentage\n");
}
```

5.1.4 Executing Code in `if` Conditions

While primarily used for conditions, C allows expressions that evaluate to a non-zero value (true) or zero (false) within the parentheses. This means you can sometimes perform assignments or function calls directly within the condition, though it's often discouraged for readability.

```
int x = 10;
if (x = 5) { // Assigns 5 to x, then evaluates to 5 (true)
    printf("x is now 5 and this code runs.\n");
}
```

5.1.5 Single Statement `if`:

If an `if` or `else` block contains only a single statement, the curly braces `{}` are optional. However, it's good practice to always use them to avoid ambiguity and potential errors when adding more statements later.

```
if (score > 90)
    printf("Excellent!\n");
else
    printf("Keep trying.\n");
```

5.1.6 switch Statement

C provides `switch` as an alternative to multiple `if-else` statements:

```
switch (variable) {
    case value1:
        // statements
        break;
    case value2:
        // statements
        break;
    default:
        // statements
        break;
}
```

Example:

```
int grade;
printf("Enter grade (1-5): ");
scanf("%d", &grade);

switch (grade) {
    case 5:
        printf("Excellent!\n");
        break;
    case 4:
        printf("Very Good!\n");
        break;
    case 3:
        printf("Good!\n");
        break;
```

```
    case 2:
        printf("Fair!\n");
        break;
    case 1:
        printf("Poor!\n");
        break;
    default:
        printf("Invalid grade!\n");
        break;
}
```

Understanding `break` and Fall-through: In C's `switch` statement, the `break` keyword is essential. If `break` is omitted from a `case` block, execution will "fall through" to the next `case` block (and subsequent ones) until a `break` is encountered or the end of the `switch` statement is reached. This "fall-through" behavior can be intentionally used for specific logic, but it's a common source of bugs if not intended.

Example of Fall-through:

```
int day = 2; // Monday
switch (day) {
    case 1:
        printf("Weekend!\n");
        break;
    case 2:
    case 3:
    case 4:
    case 5:
        printf("Weekday.\n"); // Execution falls through from case 2, 3, 4 to 5
        break;
    case 6:
        printf("Weekend!\n");
        break;
    default:
        printf("Invalid day.\n");
        break;
}
// Output for day = 2: Weekday.
```

This example shows how `case 2`, `case 3`, `case 4`, and `case 5` all execute the same `printf("Weekday.\n");` because there are no `break` statements between them.

5.2 Iteration Statements (Loops)

5.2.1 for Loop

Python vs C Syntax:

Python:

```
for i in range(5):
    print(i)

for i in range(1, 10, 2):
    print(i)
```

C:

```
// Basic for loop
for (int i = 0; i < 5; i++) {
    printf("%d\n", i);
}

// Step by 2
for (int i = 1; i < 10; i += 2) {
    printf("%d\n", i);
}
```

For Loop Structure:

```
for (initialization; condition; increment/decrement) {
    // loop body
}
```

5.2.2 while Loop

Python vs C:

Python:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

C:

```
int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

5.2.3 do-while Loop

C provides `do-while` loop (not available in Python):

```
int choice;
do {
    printf("Enter choice (1-3): ");
    scanf("%d", &choice);

    if (choice < 1 || choice > 3) {
        printf("Invalid choice! Try again.\n");
    }
} while (choice < 1 || choice > 3);
```

Key Difference: `do-while` executes the loop body at least once, even if the condition is initially false.

5.2.4 Loop Control Statements

Statement	Python	C	Purpose
`break`	`break`	`break;`	Exit loop immediately
`continue`	`continue`	`continue;`	Skip to next iteration

Example:

```
for (int i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
        continue;  // Skip even numbers
    }
    if (i > 7) {
        break;     // Stop when i > 7
    }
    printf("%d ", i);  // Output: 1 3 5 7
}
```

Revision #1
Created 2025-09-01 03:36:11 UTC by DS
Updated 2025-09-01 03:36:47 UTC by DS