C - Character Arithmetic

In C, **character arithmetic** means performing arithmetic operations like addition, subtraction, multiplication, and division on characters. Characters in C are stored as numbers using the ASCII system.

The ASCII system represents each character, digit, or symbol using numbers. **For example**, 'A' is stored as 65, 'B' as 66, 'a' as 97, and so on

Character Arithmetic Operations

To perform arithmetic operations on a character, we should know that a character takes 1 byte of memory. A signed char has a range from **-128 to 127**, and an unsigned char has a range from **0 to 255**.

Checking Character Values

Let's first see a character and its ASCII value. In the below example, we print the character '**A**' using **%c** and its ASCII value using **%d**.

```c
#include <stdio.h>
int main() {
    char ch = 'A';
    printf("Character: %c\n", ch);
    printf("ASCII Value: %d\n", ch);
    return 0;
}
```

After running the program, we can see the character and its ASCII value-

```
Character: A
ASCII Value: 65
```

Adding Two Characters

We can add characters just like numbers because each character is stored as a number.

```c
#include <stdio.h>

int main() {
    char a = 'A';
    char b = 'B';

    printf("a = %c\n", a);
    printf("b = %c\n", b);
    printf("a + b = %c\n", a + b);
    printf("a + b (as number) = %d\n", a + b);
    return 0;
}
```

Here's the **output** of the above program after adding the characters -

```
a = A
b = B
a + b = Ã¢
a + b (as number) = 131
```

In this program, the characters **a** and **b** are declared and assigned with values '**A**' and '**B**'. When we add a and b, the result is the character **Ã¢**. The ASCII value of A is 65 and B is 66, so adding them gives **65 + 66 = 131**. The character with ASCII value **131** is **Ã¢**.

Shifting and Subtracting Characters

We can shift a character forward or backward by adding or subtracting a number. **For example**, 'A' + 1 gives 'B' and 'C' - 1 gives 'B'. We can also find the difference between characters using subtraction, like 'C' - 'A', which gives 2 because the ASCII value of 'C' is 67 and 'A' is 65, so **67 - 65 = 2**.

```c
#include <stdio.h>

int main() {
    char c1 = 'A' + 1;
    char c2 = 'C' - 1;
    int diff = 'C' - 'A';

    printf("A + 1 = %c\n", c1);
    printf("C - 1 = %c\n", c2);
    printf("C - A = %d\n", diff);
    return 0;
}
```

Here's the **output** of the program after shifting and subtracting the characters -

```
A + 1 = B
C - 1 = B
C - A = 2
```

Comparing Two Characters

We can compare characters directly using comparison operators like **<**, **>**, **<=**, **>=**, **==**, and **!=** to check the order or equality of characters. In the program below, we check if the character 'a' comes before 'z'. Since the ASCII value of 'a' is 97 and 'z' is 122, so **'a' < 'z'** is true.

```c
#include <stdio.h>
int main() {
    if ('a' < 'z') {
        printf("'a' comes before 'z'\n");
    }
    return 0;
}
```

Here's the **output** that shows which character comes first.

```
'a' comes before 'z'
```

Increment and Decrement Operators on Characters

Characters in C can be incremented (**++**) or decremented (**--**) like numbers. Incrementing moves a character forward in the ASCII table, and decrementing moves it backward. In the below program, chracter **ch** starts as '**A**'. After **ch++**, it becomes '**B**', and after **ch--** it becomes '**A**' again.

```c
#include <stdio.h>

int main() {
    char ch = 'A';
    printf("Original character: %c\n", ch);

    ch++; // Increment
    printf("After increment: %c\n", ch);

    ch--; // Decrement
    printf("After decrement: %c\n", ch);
    return 0;
}
```

Following is the **output** of the above program-

```
Original character: A
After increment: B
After decrement: A
```

Converting to Uppercase and Lowercase

In ASCII, the difference between uppercase and lowercase letters is **32**. To convert an uppercase letter to lowercase, we add 32 to it. **For example**, 'M' + 32 becomes 'm'. Similarly, to convert a lowercase letter to uppercase, we subtract 32 from it. **For example**, 'm' - 32 becomes 'M'.

```
int main() {
    char upper = 'M';
    char lower = upper + 32; // Convert uppercase to lowercase

    char small = 'm';
    char capital = small - 32; // Convert lowercase to uppercase

    printf("Original Uppercase: %c\n", upper);
    printf("Converted to Lowercase: %c\n", lower);
    printf("Original Lowercase: %c\n", small);
    printf("Converted to Uppercase: %c\n", capital);
    return 0;
}
```

Here is the **output** of the above program that shows the original characters and their converted form.

```
Original Uppercase: M
Converted to Lowercase: m
Original Lowercase: m
Converted to Uppercase: M
```

In this chapter, we covered how to perform different types of **arithmetic operations on characters in C**. It is possible to perform such arithmetic operations on characters directly because the computer stores each character as a number, which allows us to add, subtract, compare and so on.

TOP TUTORIALS

Python Tutorial

Java Tutorial

C++ Tutorial

C Programming Tutorial

C# Tutorial

PHP Tutorial

R Tutorial

HTML Tutorial

CSS Tutorial

JavaScript Tutorial

SQL Tutorial

TRENDING TECHNOLOGIES

Cloud Computing Tutorial

Amazon Web Services Tutorial

Microsoft Azure Tutorial

Git Tutorial

Ethical Hacking Tutorial

Docker Tutorial

Kubernetes Tutorial

DSA Tutorial

Spring Boot Tutorial

SDLC Tutorial

Unix Tutorial

CERTIFICATIONS

Business Analytics Certification

Java & Spring Boot Advanced Certification

Data Science Advanced Certification

Cloud Computing And DevOps

Advanced Certification In Business Analytics

Artificial Intelligence And Machine Learning

DevOps Certification

Game Development Certification

Front-End Developer Certification

AWS Certification Training

Python Programming Certification

COMPILERS & EDITORS

Online Java Compiler

Online Python Compiler

Online Go Compiler

Online C Compiler