Unary operators in C/C++

Difficulty Level : Basic • Last Updated : 03 Apr, 2023

Unary operators: are operators that act upon a single operand to produce a new value.

Types of unary operators:

1. unary minus(-)
2. increment(++)
3. decrement(- -)
4. NOT(!)
5. Addressof operator(&)
6. sizeof()

Time complexity of any unary operator is O(1).

Auxiliary Space of any unary operator is O(1).

1. unary minus: The minus operator changes the sign of its argument. A positive number becomes negative, and a negative number becomes positive.

```
int a = 10;
int b = -a;  // b = -10
```

unary minus is different from the subtraction operator, as subtraction requires two operands.

Below is the implementation of **unary minus (-)** operator:

C++

```
// C++ program to demonstrate the use of 'unary minus'
// operator

#include <iostream>
using namespace std;

int main()
{
    int positiveInteger = 100;
    int negativeInteger = -positiveInteger;

    cout << "Positive Integer: " << positiveInteger << endl;
    cout << "Negative Integer: " << negativeInteger << endl;

    return 0;
}

// This code is contributed by sarajadhav12052009
```

Output

```
Positive Integer: 100
Negative Integer: -100
```

2. increment: It is used to increment the value of the variable by 1. The increment can be done in two ways:

2.1 prefix increment: In this method, the operator precedes the operand (e.g., ++a). The value of the operand will be altered *before* it is used.

```
int a = 1;
int b = ++a;  // b = 2
```

2.2 postfix increment: In this method, the operator follows the operand (e.g., a++). The value operand will be altered *after* it is used.

```
int a = 1;
int b = a++;   // b = 1
int c = a;     // c = 2
```

3. decrement: It is used to decrement the value of the variable by 1. The decrement can be done in two ways:

3.1 prefix decrement: In this method, the operator precedes the operand (e.g., – -a). The value of the operand will be altered *before* it is used.

```
int a = 1;
int b = --a;  // b = 0
```

3.2 postfix decrement: In this method, the operator follows the operand (e.g., a- -). The value of the operand will be altered *after* it is used.

```
int a = 1;
int b = a--;   // b = 1
int c = a;     // c = 0
```

4. NOT(!): It is used to reverse the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

```
If x is true, then !x is false
If x is false, then !x is true
```

Below is the implementation of the **NOT (!)** operator:

C++

```
// C++ program to demonstrate the use of '!(NOT) operator'

#include <iostream>
using namespace std;

int main()
{

    int a = 10;
    int b = 5;

    if (!(a > b))
        cout << "b is greater than a" << endl;
    else
        cout << "a is greater than b" << endl;

    return 0;
}

// This code is contributed by sarajadhav12052009
```

Output

```
a is greater than b
```

5. Addressof operator(&): It gives an address of a variable. It is used to return the memory address of a variable. These addresses returned by the address-of operator are known as pointers because they “point” to the variable in memory.

```
& gives an address on variable n
int a;
int *ptr;
ptr = &a; // address of a is copied to the location ptr.
```

Below is the implementation of **Addressof operator(&)**:

C++

```
// C++ program to demonstrate the use of 'address-of(&)'
// operator

#include <iostream>
using namespace std;

int main()
{

    int a;
    int* ptr;

    ptr = &a;

    cout << ptr;

    return 0;
}

// This code is contributed by sarajadhav12052009
```

Output

```
0x7ffddcf0c8ec
```

6. sizeof(): This operator returns the size of its operand, in bytes. The *sizeof()* operator always precedes its operand. The operand is an expression, or it may be a cast.

Note: The **`sizeof()`** operator in C++ is machine dependent. For example, the size of an ‘int’ in C++ may be 4 bytes in a 32-bit machine but it may be 8 bytes in a 64-bit machine.

Below is the implementation of **sizeof()** operator:

C++

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

int main()
{
    float n = 0;
    cout << "size of n: " << sizeof(n);
    return 0;
}
```

Output

```
size of n: 4
```

167

Related Articles

1. Operators in C | Set 2 (Relational and Logical Operators)
2. What are the Operators that Can be and Cannot be Overloaded in C++?
3. Increment (Decrement) operators require L-value Expression
4. Order of operands for logical operators
5. Conversion Operators in C++
6. const_cast in C++ | Type Casting operators
7. How to sum two integers without using arithmetic operators in C/C++?
8. Execution of printf With ++ Operators in C
9. Conditionally assign a value without using conditional and arithmetic operators
10. new and delete Operators in C++ For Dynamic Memory