C - Misc Operators

Besides the main categories of operators (arithmetic, logical, assignment, etc.), C uses the following operators that are equally important. Let us discuss the operators classified under this category.

The "&" symbol, already defined in C as the **Binary AND Operator** copies a bit to the result if it exists in both operands. The "&" symbol is also defined as the **address−of operator.**

The "*" symbol − A well−known arithmetic operator for multiplication, it can also be used as a **dereference operator.**

C uses the ">" symbol, defined as a **ternary operator,** used to evaluate a conditional expression.

In C, the dot "." symbol is used as the member access operator in connection with a struct or union type.

C also uses the arrow "→" symbol as an indirection operator, used especially with the pointer to the struct variable.

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
?:	Conditional Expression.	If Condition is true ? then value X, else value Y
.	Member access operator	var.member
−>	Access members of a struct variable with pointer	ptr −> member;

The sizeof Operator in C

The sizeof operator is a compile−time unary operator. It is used to compute the size of its operand, which may be a data type or a variable. It returns the size in number of bytes. It can be applied to any data type, float type, or pointer type variables.

```
sizeof(type or var);
```

When sizeof() is used with the data types, it simply returns the amount of memory allocated to that data type. The output can be different on different machines like a 32−bit system can show different output while a 64−bit system can show different of the same data types.

Example

Here is an example in C language

</> Open Compiler

```
#include <stdio.h>

int main(){

   int a = 16;

   printf("Size of variable a : %d\n",sizeof(a));
   printf("Size of int data type : %d\n",sizeof(int));
   printf("Size of char data type : %d\n",sizeof(char));
   printf("Size of float data type : %d\n",sizeof(float));
   printf("Size of double data type : %d\n",sizeof(double));

   return 0;
}
```

Output

When you run this code, it will produce the following output −

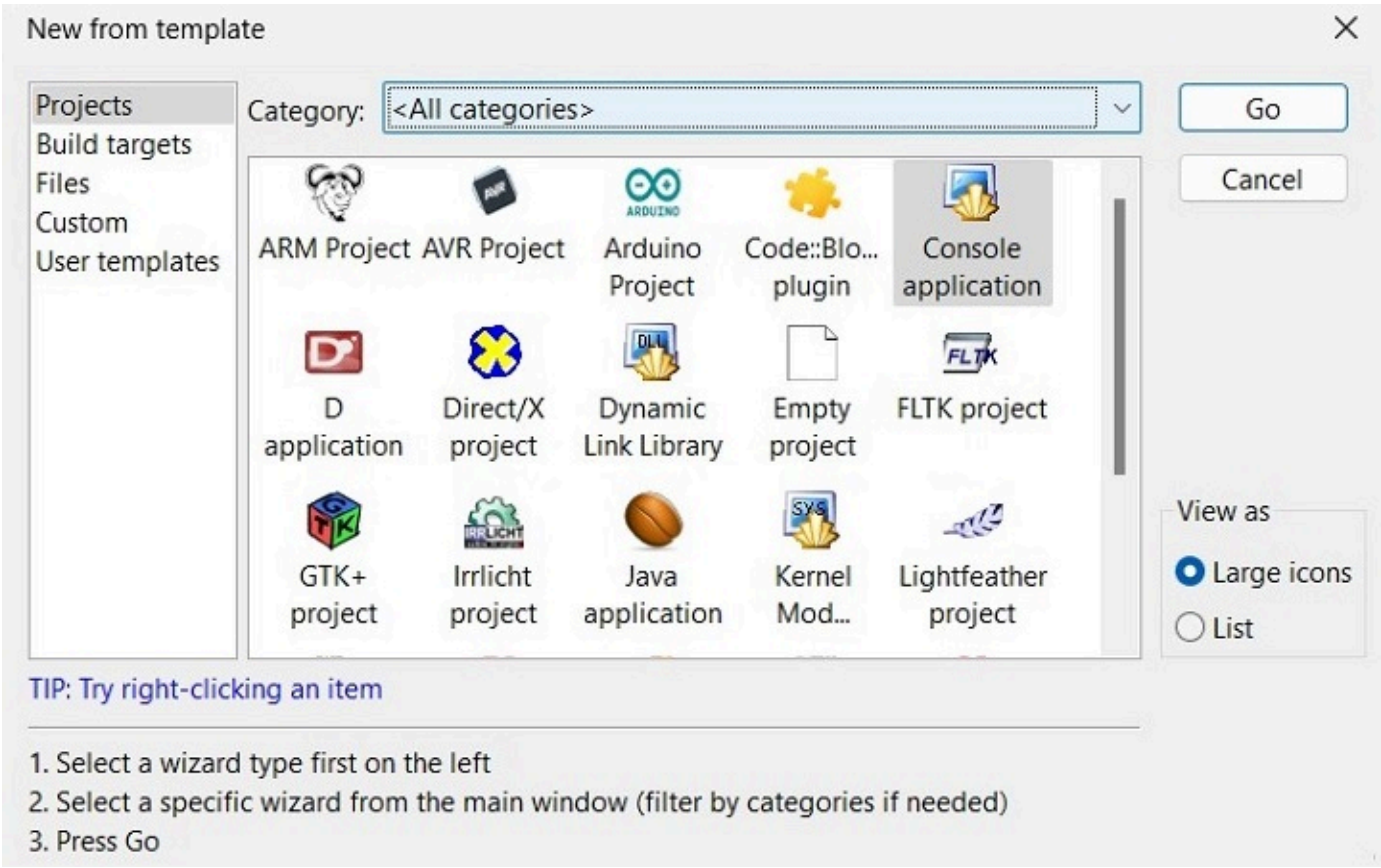
```
Size of variable a: 4
Size of int data type: 4
Size of char data type: 1
Size of float data type: 4
Size of double data type: 8
```

Address-of Operator in C

The "&" operator returns the address of an existing variable. We can assign it to a pointer variable –

```
int a;
```

Assuming that the compiler creates the variable at the address 1000 and "**x**" at the address 2000, then the address of "**a**" is stored in "**x**".



Example

Let us understand this with the help of an example. Here, we have declared an **int** variable. Then, we print its value and address –

</> Open Compiler

```
#include <stdio.h>

int main(){

   int var = 100;

   printf("var: %d address: %d", var, &var);
```

```
   return 0;
}
```

Output

Run the code and check its output –

```
var: 100 address: 931055924
```


The Dereference Operator in C

To declare a pointer variable, the following syntax is to be used –

```
type *var;
```

The name of the variable must be prefixed with an asterisk (*). The data type indicates it can store the address of which data type. For example –

```
int *x;
```

In this case, the variable x is meant to store the address of another **int** variable.

```
float *y;
```

The **"y"** variable is a pointer that stores the memory location of a float variable.

The **"&"** operator returns the address of an existing variable. We can assign it to the pointer variable –

```
int a;
int *x = &a;
```

We can see that the address of this variable (any type of variable for that matter) is an integer. So, if we try to store it in a pointer variable of int type, see what happens –

```
float var1 = 10.55;
int *intptr = &var1;
```

The compiler doesn't accept this, and reports the following error –

```
initialization of 'int *' from incompatible pointer type 'float *' [-Wincompatible
```

It indicates that the type of a variable and the type of its pointer must be the same.

In C, variables have specific data types that define their size and how they store values. Declaring a pointer with a matching type (e.g., **"float *"**) enforces type compatibility between the pointer and the data it points to.

Different data types occupy different amounts of memory in C. For example, an int typically takes 4 bytes, while a float might take 4 or 8 bytes depending on the system.

Adding or subtracting integers from pointers moves them in memory based on the size of the data they point to.

Hence, we declare the **floatptr** variable of float * type.

Example 1

Take a look at the following example –

</> Open Compiler

```
#include <stdio.h>

int main(){

   float var1 = 10.55;
   float *floatptr = &var1;

   printf("var1: %f \naddress of var1: %d \n\nfloatptr: %d \naddress of floatptr:

   return 0;
}
```

Output

```
var1: 10.550000
address of var1: 6422044

floatptr: 6422044
address of floatptr: 6422032
```

Example 2

The * operator is called the Dereference operator. It returns the value stored in the address which is stored in the pointer, i.e., the value of the variable it is pointing to. Take a look at the following example −

</> Open Compiler

```
#include <stdio.h>

int main(){

   float var1 = 10.55;
   float *floatptr = &var1;

   printf("var1: %f address of var1: %d\n",var1, &var1);
   printf("floatptr: %d address of floatptr: %d\n", floatptr, &floatptr);
   printf("var1: %f value at floatptr: %f", var1, *floatptr);

   return 0;
}
```

Output

On running this code, you will get the following output −

```
var1: 10.550000 address of var1: 6422044
floatptr: 6422044 address of floatptr: 6422032
var1: 10.550000 value at floatptr: 10.550000
```

The Ternary Operator in C

In C language, the "?" character is used as the ternary operator. It is also known as a **conditional operator.**

The term "ternary" implies that the operator has three operands. The ternary operator is often used to put conditional (if−else) statements in a compact way.

The ? operator is used with the following **syntax** −

```
exp1 ? exp2 : exp3
```

It has the following **three operands** −

- **exp1** − a Boolean expression that evaluates to True or False
- **exp2** − returned by the ? operator when **exp1** is true
- **exp3** − returned by the ? operator when **exp1** is false

Example

The following C program uses the ? operator to check if the value of a is even or odd.

</> Open Compiler

```
#include <stdio.h>

int main(){

   int a = 10;
   (a % 2==0) ? printf("%d is Even\n", a) : printf("%d is Odd\n", a);

   return 0;
}
```

Output

```
10 is Even
```

Change the value of "a" to 15 and run the code again. Now you will get the following output −

```
15 is Odd
```

The conditional operator is a compact representation of if − else construct.

The Dot (.) Operator in C

In C language, you can define a derived data type with struct and union keywords. A derived or user−defined data type that groups together member elements of different types.

The dot operator is a **member selection operator,** when used with the struct or union variable. The dot (.) operator has the **highest operator precedence in C** Language and its associativity is from left to right.

Take a look at its **syntax** –

```
var.member;
```

Here, var is a variable of a certain struct or a union type, and member is one of the elements defined while creating structure or union.

A new derived data type is defined with struct keyword as following syntax –

```
struct newtype {
   type elem1;
   type elem2;
   type elem3;
   . . .
   . . .
};
```

You can then declare a variable of this derived data type as –

```
struct newtype var;
```

To access a certain member,

```
var.elem1;
```

Example

Let us declare a struct type named book, declare a struct variable. The following example shows the use of "." operator to access the members in the book structure.

</> Open Compiler

```
#include <stdio.h>

struct book{
   char title[10];
   double price;
   int pages;
};
```

```
int main(){

   struct book b1 = {"Learn C", 675.50, 325};
   printf("Title: %s\n", b1.title);
   printf("Price: %lf\n", b1.price);
   printf("No of Pages: %d\n", b1.pages);
   printf("size of book struct: %d", sizeof(struct book));

   return 0;
}
```

Output

On running this code, you will get the following output –

```
Title: Learn C
Price: 675.500000
No of Pages: 325
size of book struct: 32
```

The Indirection Operator in C

A structure is a derived data type in C. In C, the struct keyword has been provided to define a custom data type.

A new derived data type is defined with a **struct** keyword as the following **syntax** –

```
struct type {
   type var1;
   type var2;
   type var3;
   . . .
   . . .
};
```

You can then declare a variable of this derived data type as –

```
struct type = var;
```

Usually, a struct is declared before the first function is defined in the program, after the include statements. That way, the derived type can be used for declaring its variable inside

any function.

Let us declare a struct type named book as follows −

```
struct book {
   char title[10];
   double price;
   int pages;
};
```

To declare a variable of this type, use the following syntax −

```
struct book b1;
```

The initialization of a struct variable is done by placing value of each element inside curly brackets.

```
struct book b1 = {"Learn C", 675.50, 325};
```

You can also store the address of a struct variable in the struct pointer variable.

```
struct book *strptr;
```

To store the address, use the "&" operator.

```
strptr = &b1;
```

C defines the arrow (→) symbol to be used with struct pointer as indirection operator (also called struct dereference operator). It helps to access the elements of the struct variable to which the pointer reference to.

Example

In this example, strptr is a pointer to struct book b1 variable. Hence, strrptr−>title returns the title, similar to b1.title does.

</> Open Compiler

```
#include <stdio.h>
#include <string.h>

struct book {
   char title[10];
```

```
   double price;
   int pages;
};

int main() {
   struct book b1 = {"Learn C", 675.50, 325};
   struct book *strptr;
   strptr = &b1;
   printf("Title: %s\n", strptr->title);
   printf("Price: %lf\n", strptr->price);
   printf("No of Pages: %d\n", strptr->pages);

   return 0;
}
```

Output

Run the code and check its output –

```
Title: Learn C
Price: 675.500000
No of Pages: 325
```