Type Casting is basically a process in C in which we change a variable belonging to one data type to another one. In type casting, the compiler automatically changes one data type to another one depending on what we want the program to do. For instance, in case we assign a float variable (floating point) with an integer (int) value, the compiler will ultimately convert this int value into the float value. The process of casting allows the programmers to make such types of conversions explicit or even force it in cases where it wouldn’t happen normally.

**What is Type Casting in C?**

Type casting is the process in which the compiler automatically converts one data type in a program to another one. Type conversion is another name for type casting. For instance, if a programmer wants to store a long variable value into some simple integer in a program, then they can type cast this long into the int. Thus, the method of type casting lets users convert the values present in any data type into another data type with the help of the cast operator, like:

(type\_name) expression

Let us take a look at the following example to understand how we can utilise the cast operator for dividing an integer variable with another one by performing it as an operation of floating-point type:

#include <stdio.h>

main() {

int total = 17, values = 5;

double average;

average = (double) total / values;

printf(“The average of all the values available with us is : %f\n”, average );

}

The compilation and execution of the code mentioned here would generate the following output of the program:

The average of all the values available with us is : 3.400000

You must note here that the precedence of the cast operator is much higher than that of the division operator. Thus, the program would first convert the value of *total*into the*double*type. After this, it will finally divide this value with the help of count yielding of the double value.

As a matter of fact, the process of type conversion or type casting can be very implicit. It means that the compiler is capable of performing it automatically. Conversely, we can specify the data type explicitly using the cast operator. Both the methods are okay, but using the cast operator whenever in need is considered a better programming practice (explicit).

**Syntax for Type Casting in C**

int val\_1;

float val\_2;

// Body of program

val\_2 = (float) val\_1; // type casting of the values

**Types of Type Casting in C**

The process of type casting can be performed in two major types in a C program. These are:

* Implicit
* Explicit

**C Language Implicit Type Casting**

Implementing the implicit type casting is very easy in a program. We use it to convert the data type of any variable without losing the actual meaning that it holds. The implicit type casting occurs automatically.

In simpler words, the implicit type casting performs the conversion without altering any of the values stored in the program’s variables.

Follow these points to understand what rules the implicit type casting follows in a C program:

* If we are performing a conversion on two of the different data types in a program, then the conversion of the lower data type to the higher data type will occur automatically.
* If we suppose that we are performing the type casting operation among two different data types like float and int, then the resultant value would be the floating data type (float).

**Examples of Implicit Type Casting in C**

int x = 4;

float a = 12.4, b;

b = a / x;

In the example given above, the variable x will get converted to the float data type (float) automatically, and it is comparatively a bigger data type in C. Here, the variable x and the variable a would be equal in terms of their data types. Thus, the value of b would be equal to 12.4/4.0=3.1.

**The Conversion of Char to Int**

We can use the process of type casting to convert the character (char) data type to the int (integer) data type in C. When we are performing a conversion between these two, the resultant value would be an integer (int) data type. It is because the int data type is comparatively bigger than the char data type in C.

When this conversion occurs, firstly, the value of the char data type gets converted to its corresponding ASCII value. After this, it gets processed further accordingly.

Example

#include <stdio.h>

int main() {

int val = 10;

char x = ‘T’;

int total;

total = val + x;

printf(“In the code, we convert the char data type to int data type\n”);

printf(“The total of all these values is : %d\n”, total);

return 0;

}

The output obtained from the code mentioned above would be:

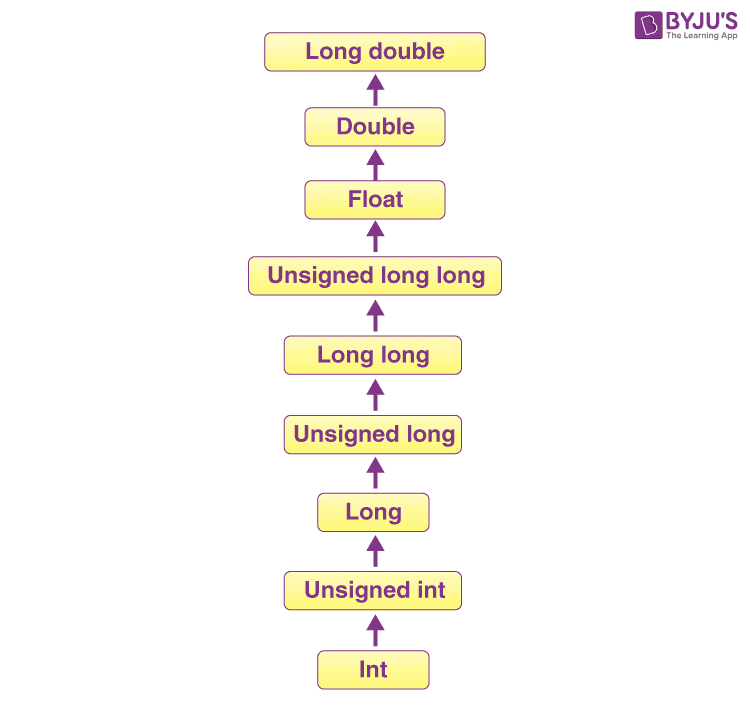
In the code, we convert the char data type to int data type

The total of all these values is : 94

In the example mentioned above, the compiler would convert the value allotted to the char x variable to its corresponding ASCII value. Here, the ASCII value of the char x = T is equivalent to 84. Thus, the value of*total*would be equal to the sum of the numbers 10 and 84. At last, 94 would be printed as an output of the code.

**Hierarchy of Arithmetic Conversion**

In this case, the compiler will perform the process of integer promotion first. After this, it will check if two of the operands consist of different data types. In case they both have varied data types, then the conversion will occur in the hierarchy as follows:



Example

#include <stdio.h>

int main() {

int val = 21;

char xyz = ‘T’;

float total;

total = val + xyz;

printf(“Let us look at the case of Arithmetic Conversion Hierarchy with Type Casting\n”);

printf(“The total of all these values is equivalent to: %f\n”, total);

}

The output of the code mentioned above would be as follows:

Let us look at the case of Arithmetic Conversion Hierarchy with Type Casting.

The total of all these values is equivalent to: 105.000000

In the example mentioned above, the*xyz*variable gets converted to the equivalent ASCII value. But here, the compiler converts the*val*and*xyz*into the floating data type (float). After adding both of these, a float data type will be generated as the output by the compiler.

**C Language Explicit Type Casting**

This process is not at all like the implicit type casting in C, where the conversion of data type occurs automatically. Conversely, in the case of explicit type casting, the programmer needs to force the conversion. In simpler words, one has to perform type casting on the data types on their own.

**Syntax:**

(name\_of\_data\_type) expression

Here, name\_of\_data\_type refers to the name of that data type to which we want to convert the available data type in the code. This expression can be a constant, a variable, or even an actual expression in a program.

**Examples of Explicit Type Casting in C**

#include<stdio.h>

int main()

{

float num = 56.3;

int p = (int)num + 50; // data type casting explicitly

printf(“Let us understand Explicit Type Casting in C\n”);

printf(“The value of the digit used is: %f\n”, num);

printf(“The value of the variable p is: %d\n”,p);

return 0;

}

The output obtained by the code mentioned above would be as follows:

Let us understand Explicit Type Casting in C

The value of the digit used is: 56.299999

The value of the variable p is: 106

In the example mentioned above, we perform the declaration of a variable of a float type, named *num*. After this, the float variable gets converted into an int (integer) data type with the help of explicit data type casting. After this, it gets added with the number 50, and we get an output of an integer.

**Integer Promotion**

We can use integer promotion to convert those integer type values that are smaller than the unsigned integer (unsigned int) or integer (int) to the unsigned int or int values.

Example

#include <stdio.h>

int main(){

int digit = 21;

char a = ‘T’;

int total;

printf(“Let us look at Integer Promotion in C\n”);

total = digit + a;

printf(“The total of all the digits is equal to: %d\n”, total);

return 0;

}

The output obtained out of the program mentioned above would be:

Let us look at Integer Promotion in C

The total of all the digits is equal to: 105

In the example mentioned above, the value generated for*total*would be equal to the number 105. Here, the compiler is performing the integer promotion first, then converting the value allotted to a (T) to its corresponding ASCII value, and then adding the value of the*digit*variable to it.

**C Language In-built Data Type Casting Functions**

There are basically 5 different in-built type casting functions available in the C language. These are:

* atof(): We use it to convert the data type string into the data type float.
* atoi(): We use it to convert the data type string into the data type int.
* atbol(): We use it to convert the data type string into the data type long.
* itoba(): We use it to convert the data type int into the data type string.
* ltoa(): We use it to convert the data type long into the data type string.

**The Differences Between Type Conversion and Type Casting in C**

Though both of these terms are often used interchangeably – there is a significant difference between the process of type conversion and type casting. Here is how:

* In case the conversion of one data type to another one occurs without any need for human interaction, it is referred to as type conversion. On the other hand, whenever a user performs the conversion of any data type to another one, it is referred to as type casting in C.
* One has to make use of the ‘()’ operator in the case of type casting, but there is no need for any operator in the case of type conversion in C language.
* The process of type casting occurs when a program gets written. On the other hand, we perform type conversion during the process of compilation.
* Type casting is mainly used when both the data types in the equation aren’t compatible at all with each other. On the other hand, it is a must that both data types be compatible with each other in the case of type conversion.

**Type Casting Advantages in the C Language**

Type casting offers the following advantages in the C language:

* Type casting helps a programmer convert any given data type to any other data type in a program.
* It makes any given program very lightweight.
* Using type casting, one can take advantage of several features, such as arithmetic conversion hierarchy and type representations.

**Practice Problems on Type Casting in C**

**1.**What would be the output obtained out of the program mentioned below2?

#include <stdio.h>

main() {

int a = 17;

char b = ‘c’;

int total;

total = a + b;

printf(The addition of both the integers would return the value : %d\n”, total );

}

**A.**The addition of both the integers would return the value : 187

**B.**The addition of both the integers would return the value : 54

**C.**The addition of both the integers would return the value : 93

**D.**The addition of both the integers would return the value : 116

**Answer:** **D.**The addition of both the integers would return the value: 116

**2.**What would be the output obtained out of the program mentioned below?

#include <stdio.h>

main() {

int p = 17;

char q = ‘A’;

float total;

total = p + q;

printf(“The total of both the digits is equal to : %f\n”, total );

}

**A.**The total of both the digits is equal to : 95.000000

**B.**The total of both the digits is equal to : 82.000000

**C.**The total of both the digits is equal to : 176.000000

**D.**The total of both the digits is equal to : 34.000000

**Answer:** **B.**The total of both the digits is equal to : 82.000000