Type Conversions

Learn about converting one data type to another through the implicit and the explicit type conversions.

We'll cover the following

There are two kinds of type conversion we need to talk about: automatic or **implicit** type conversion and **explicit** type conversion.

Implicit Type Conversion

The operators we have looked at can deal with different types. For example, we can apply the addition operator `+` to an `int` as well as a `double`. It is important to understand how operators deal with different types that appear in the same expression. There are rules in C that govern how operators convert different types, to evaluate the results of expressions.

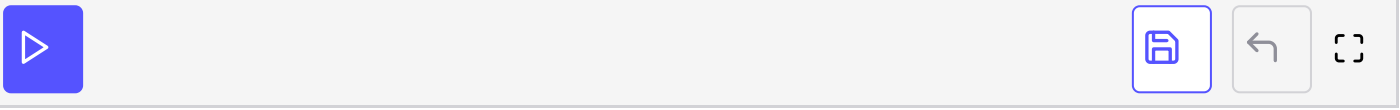
For example, when a floating-point number is assigned to an integer value in C, the decimal portion of the number gets truncated. On the other hand, when an integer value is assigned to a floating-point variable, the decimal is assumed as `.0`.

This sort of implicit or automatic conversion can produce nasty bugs that are difficult to find, especially for example when performing multiplication or division using mixed types, e.g., integer and floating-point values. Here is some example code illustrating some of these effects:

```
#include <stdio.h>
int main() {
    int a = 2;
    double b = 3.5;
    double c = a * b;
    double d = a / b;
```

```
    int e = a * b;
    int f = a / b;
    printf("a=%d, b=%.3f, c=%.3f, d=%.3f, e=%d, f=%d\n",

            a, b, c, d, e, f);
    return 0;
}
```

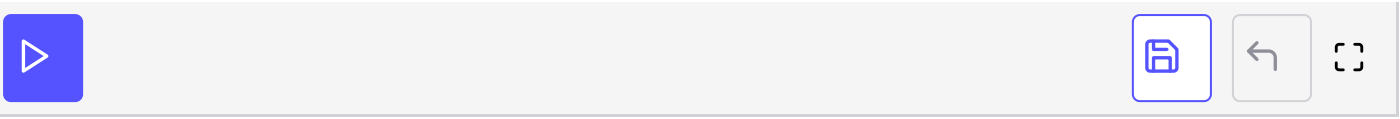


Explicit Type Conversion

Type Casting

There is a mechanism in C to perform **type casting**, that is to force an expression to be converted to a particular type of our choosing. We surround the desired type in brackets and place that just before the expression to be coerced. Look at the following example code:

```
#include <stdio.h>
#include <stdio.h>
int main() {
    int a = 2;
    int b = 3;
    printf("a / b = %.3f\n", a/b);
    printf("a / b = %.3f\n", (double) a/b);
    return 0;
}
```



After running the code, you will get a warning about line 6. This implies that %f specifier expects a double or float value but instead a/b is an integer value. The compiler does not warn about line 7 as we have type casted the integer type value to a double type value.

String Conversion Library Functions

There are some built-in library functions in C to perform some basic conversions between strings and numeric types. Two useful functions to know about converting ASCII strings to numeric types: `atoi()` (ASCII to an integer) and `atof()` (ASCII to floating-point). We need to `#include` the library `stdlib.h` to use these functions.

Converting numeric types to strings is a bit more difficult. First, we have to allocate space in memory to store the string. Then we use the `sprintf()` built-in function to “print” the numeric type into our string.

Here is some example code illustrating conversion of strings to numerics, and vice-versa:

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

int main() {
    char intString[] = "1234";
    char floatString[] = "328.4";
    int myInt = atoi(intString);
    double myDouble = atof(floatString);
    printf("intString=%s, floatString=%s\n", intString, floatString);
    printf("myInt=%d, myDouble=%.1f\n\n", myInt, myDouble);

    int a = 2;
    double b = 3.14;
    char myString1[64], myString2[64];
    sprintf(myString1, "%d", a);
    sprintf(myString2, "%.2f", b);
    printf("a=%d, b=%.2f\n", a, b);
    printf("myString1=%s, myString2=%s", myString1, myString2);
    return 0;
}
```

Other than the prebuilt data types provided to us, C also allows us to define our customized data types. Find out more in the next lesson.