

## Overview

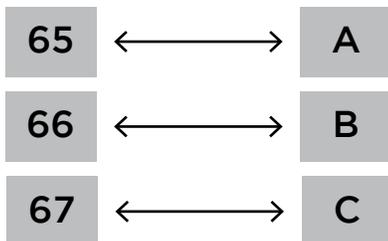
Recall that C has several different data types, including **ints**, **floats**, and **chars**. It may sometimes be necessary to convert variables from one data type to another data type. C allows us to do this via **typecasting** (or just "casting"). Typecasting allows you to cast data from one type to another type which is equally or less precise, but you cannot cast data from a type that is less precise to a type that is more precise.

### Key Terms

- typecasting
- explicit typecasting
- implicit typecasting

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int x = 65;
6     printf("%i\n", x); // 65
7     printf("%c\n", (char) x); // A
6 }
```



## Chars and Ints

The ASCII standard, as you may recall, gives every letter a unique number to identify it: capital A is represented by the number 65, capital B by 66, and so on. Using typecasting, we can convert between integer values and **char** values.

Say, for instance, that we assigned an integer variable **x** to hold the value **65**. If we were to print the variable out on the screen (like on line 6 of the code to the left), then it would display the number **65** to the console.

On line 7, however, we've included a placeholder for a **char** instead of an **int** (as denoted by the **%c** symbol). We're still passing in **x** as an argument, but the code first casts **x** into a **char**. This is done by writing **(char)** in parentheses before the name of the variable. Placing a new type name in front of an existing variable to evaluate the variable as a different type is called **explicitly typecasting**: we are directly providing instructions to convert types.

While explicit typecasting in this situation is good practice from a style perspective (so that people reading your code can better understand what's happening), it's not actually necessary. If we were to exclude the **(char)** symbol from before the **x** in line 7 of the above code, the code will still print out the letter **A** (the ASCII mapping of the value **65**). Since we've included a placeholder for a **char**, the compiler is expecting a **char** to be passed in. If we pass an **int** in, the compiler will automatically try to interpret the value as a **char** instead. This is called **implicit typecasting**.

## Ints and Floats

Typecasting is also valuable for converting between floating-point numbers and integers. Take the example at right. On line 6, we might want **b** to store the value of 28 divided by 5, which is 2.4. But line 6 actually sets **b** to be **2.0**. This is because the compiler sees a division between two **ints**, and thus presents the answer as an **int**, even though we're storing the value inside of a **float**. To get around this, we can first explicitly cast **a** to be a float, and then perform the division, as is done on line 7. In this case, **c** now correctly equals **2.4**.

Implicit typecasting can also be valuable when dealing with **ints** and **floats**. Since **ints** do not store digits past the decimal point, typecasting a **float** to an **int** is an easy way to truncate a number into an integer. On line 10 to the right, when we try to assign an **int** to be the floating-point value **d**, **d** is implicitly cast to be an **int**, getting rid of everything after the decimal point. The value of **e** is now **28**.

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a = 28;
6     float b = a / 5;
7     float c = (float) a / 5;
8
9     float d = 28.523;
10    int e = d;
11 }
```