Casting | Coursera

Casting

Casting

Sometimes, the programmer wants to explicitly request a conversion from one type to another—either because the compiler has no reason to insert it automatically (the types are already the same, but a different type of operation is desired), or because the compiler does not consider the conversion “safe” enough to do automatically. This explicitly requested conversion is called casting and is written in the code by placing the desired type in parentheses before the expression whose value should be converted. For example, **(double)x** evaluates x (by reading the value in its box), then converts it to a double.

To see why we would want this capability, let us begin with a seemingly benign example. We want to write a program that calculates how many hours you would work per day if you stretched the 40 hour work week across 7 days instead of 5. A naïve implementation of the code might begin with two ints, nHours and nDays.

```
int main(void) {
  int nHrs = 40;
  int nDays = 7;

  float avg = nHrs/nDays;
  printf("%d hours in %d days\n", nHrs, nDays);

  printf("work %.1f hours per day!\n", avg);

  //...

}
```

Here, int is a perfectly reasonable type as we are working only in integer numbers of hours (40) and days (5). This code then divides the number of hours by the number of days and stores the result in the float avgWorkDay. If you execute this code carefully by hand, you will find that when it prints the answer out, it will print 5.0. Somehow our work week just got shortened to 35 hours!

In this case, the problem lies in the fact that we divided two ints, and integer division always produces an integer result—in this case 5. When the compiler looks at this expression, there are only integers involved, so it sees no need to convert either operand to any other type. It therefore generates instructions that request the CPU to perform integer division, producing an integer result.

However, when the compiler examines the assignment, it sees that the type on the left (the type of the box it will store the value in) is float, while the type of the expression on the right (the type of the value that the division results in) is int. It then inserts the type conversion instruction at the point of the assignment: converting the integer division result to a floating point number as it puts it in the box.

Here, what we really wanted to do was to convert both operands to a real type (float or double) before the division occurs, then perform the division on real numbers. We can achieve this goal by introducing an explicit cast—requesting a type conversion.

```
9

6

7

8

3

4

5

1

2
```

```
}

  printf("%d hours in %d days\n", nHrs, nDays);

  printf("work %.1f hours per day!\n", avg);

  //...

  int nDays = 7;


  float avg = nHrs/(float)nDays;

int main(void) {

  int nHrs = 40;
```

We could explicitly cast both operands, however, casting either one is sufficient to achieve our goal. Once one operand is converted to a real type, the compiler is forced to automatically convert the other. We prefer writing a / (double) b over (double) a/ b even though they are the same, as the former does not require the reader of the code to remember the relative precedence ("order of operations") between a cast and the mathematical operator. However, we note that casting has very high operator precedence—it happens quite early in the order of operations. This example is shown in the following video.

✓

Completed