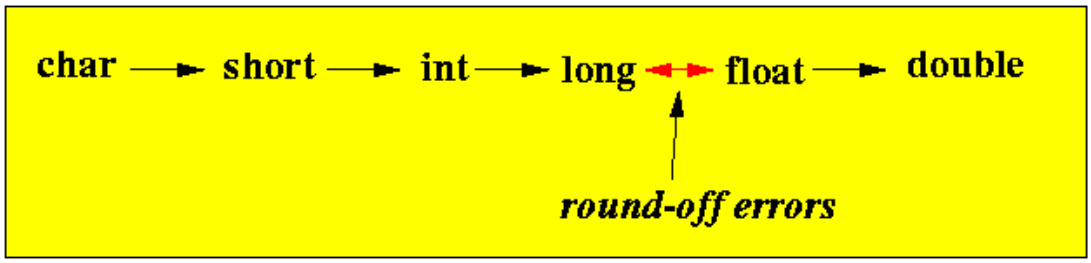
Conversion rules of C

- ***Safe* conversions in C**
  - **Safe conversion:**

    - A **safe** conversion from one data type to another data type is a **conversion** that does ***not*** cause an ***overflow***

      (I.e., you will get a **representation** for the ***same* value**)

  - The following **conversions** are **safe** in **C**:



Important Note:

- In **C**, ***safety*** has ***nothing*** to do with ***permission*** !!!

This will be explained in the **automatic conversion rule** next....

- **Automatic conversion rules in C**
  - ***Automatic* conversion rules** of **C**:

    - A ***binary* operation** using values of **2 different types**:

      - **C** will **automatically** perform a ***safe* conversion** for the value of the ***less capable* data type** to a value of the ***higher capable* data type**

        Example:

```
int a;
float b;

a + b
                1. first convert A to float
                2. then perform A + B as float + float
```

```
(Just like Java)
```

- An ***assignment*** **operation** using values of **2 different types**:

  - **C** will **automatically** perform a **conversion** for the value of the ***LHS*** **data type** to a value of the ***RHS*** **data type**

    Even if the **conversion** will cause **precision loss** (i.e., ***unsafe***) !!!

    Example:

```
int   a;
float b


a = b;
                1. first convert b to int !!!
                2. then store the value in variable a
                (DIFFERENT from Java !!!)
```

    Note: this is **forbidden** in **Java** !!! (Need a **casting** operator)

Warning:

- In **C**, you can perform ***unsafe*** **assignment operations** that Java will **not allow** without using a **casting operation**

  Rationale:

  - **C** is designed for **system programming**; and **system programmers** are ***mature*** **programmers** who ***should*** **know the danger** of all features of the programming language.
  - Furthermore, **preventive features** in **programming languages** can **prevent programmers** from performing certain tasks.
    - A **systems programming language** is **intended** to give the programmer to **complete control** of the computer (no restrictions)

    Compiler checks for type conflicts often **limit** the programmers' ability to perform some tasks.

    Hence, these checks are **omitted**.

- **Conclusion:**

- **C** is ***less*** **strongly typed** than **Java**

  (In other words: **grow up** !!!)

- **Example:**

```
#include <stdio.h>

int main(int argc, char* argv[] )
{
   int i;
   short s;

   i = 9827563;
   s = i;        /* Unsafe conversion, allowed in C !!! */

   printf( "i = %d , s = %d \n", i, s );
}
```

Output:

```
 i = 9827563 , s = -2837    (lost of accuracy !)
```

- **Example Program:** (Demo above code)

  - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `gcc casting1.c`
- To run: `./a.out`

- **Postlude**
  - **Fact:**
    - **C will *warn*** you about ***some*** **conversion** between ***different*** **data types**
      - When the **data types** are ***too far*** **apart** (i.e., ***completely*** **unrelated**, **C** will **give** a **warning**

  - **Example:**

```
int main(int argc, char* argv[] )
{
   int i = 0;  // Integer
               // i is a number that you can add, subtract, etc
   int a[5];   // Array of integers
               // a is the LOCATION (address) of the first elem of the array


   i = a;      // WARNING !! Too different !!!

   printf("i = %d\n", i);

}
```

Compiler message:

```
casting2.c: In function 'main':
casting2.c:11: warning: assignment makes integer from pointer without a cast
```

- **Example Program:** (Demo above code) *Example*
  - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `gcc casting2.c`
- To run: `a.out`
