Type Conversions

Coercion Related Problem

Some assignment with type coercion are acceptable.

```
i = 845.24; // i has 845
i = -845.24; // i has -845
```

Other cases of assignment with type coercion may not be acceptable due to range violation.

```
c = 10000;  // WRONG: exceeds range
i = 1.0e34; // WRONG: results in overflow in conversion
f = 1.0e34; // WRONG: results in loss of precision
```

Type Conversions

Example 12 (Casting)

- Use explicit type casting when needed:
  (type) expression
- Some example are given below:

```
float frac_part = f - (int) f;

i = (int) f;

float quotient;
int dividend, divisor;

quotient = (float) dividend / (float) divisor;
quotient = (float) dividend / divisor;
quotient =  dividend / (float) divisor;
```

Type Conversions

Example 13 (Casting)

- Explicit casting prevents arithmetic overflow.

```
int i;
int j = 1000;

i = j * j;           // could result in overflow
i = (long) j * j;    // prevents overflow
i = (long) (j * j); // overflow as it occurs by the time of casting
```

Logical Expressions

Relational & Equality Operators

- Relation operators: $<, <=, >=, >$
- Precedence of relation operators are less than arithmetic operators.
- `i+1 > j+3` evaluates to 1, if the relation is **true**, otherwise **false**.
- Expression `i < j < k` is equivalent to `(i < j) < k`. So, results in `1`, when `i = 10, j = 5, k = 40`.
- Equality operators `==, !=` have lower precedence, eg. `i < j == j < k` is equvalent to `(i < j) == (j < k)`

Logical Expressions

Logical Operators

- Three operators: negation (!), and (&&), (||).
- ! has same precedence as unary operators and right associative, ie., `!!a` is `!(!a)`.
- Precedence of || and && are lower than relational and equality opertor and they are left associative.
- So, `i < j && k == m` means `(i < j) && (k==m)`
- `i < j < k` is legal but meaningless.
  - Consider `i = 40, j = 160, j = 90`
  - Evaluation order: `(i < j) < k`, so it returns `true`.

Logical Expressions

Evaluation of Logical Expressions

```
int i = 2, j = 3;
int k = i * j == 6; // k = ((i * j) == 6)          => 1

int i = 5, j = 10, k = 1;
printf("%d", k > i < j); // (k > i) < j           => 1

int i = 3, j = 4, k =5;
printf("%d", i % j + i < k); // ((i % j) + i) < k => 0

int i = 10, j = 5;
printf("%d", !i < j); // (!i) < j                 => 1

int i = 5, j = 0, k = -5;
printf("%d", i && j || k); // (i && j) || (k)     => 1
```

Logical Expressions

Example 14

```
#include <stdio.h>
main() {
   int a = 1, b = 0, c = 1;

   printf("a && b is \n%i\n\n", a && b);
   printf("a && c is \n%i\n\n", a && c);
   printf("a || b is \n%i\n\n", a || b); // b not evaluated
   printf("a || c is \n%i\n\n", a || c); // c not evaluated
   printf("!a is \n%i\n\n", !a);
   printf("!b is \n%i\n\n", !b);
   printf("!(a && b) is \n%i\n\n", !(a && b));
   printf("!(a && c) is \n%i\n\n", !(a && c));
   printf("!a || !b is \n%i\n\n", !a || !b);
   printf("!a || !c is \n%i\n\n", !a || !c);
   printf("The value of a || (!c && (!b || a)) is \n%i\n\n",
     a || (!c && (!b || a))); // only a is evaluated
}
```

Logical Expressions

Example 15 (boolean values)

```
#include <stdio.h>
int main() {
    int k;

    printf("Enter k : ");
    scanf("%d", &k);

    if (k)
        printf("TRUE, value of k = %d\n", k);
    else
        printf("FALSE, value of k = %d\n", k);
}
```

Logical Expressions

Example 16

```
#include <stdio.h>
int main() {
    int i = -3, j = 2,  k = 0, m;
    m = ++i || ++j && ++k; // (++i) || ((++j) && (++k))
    printf("%d %d %d %d\n", i, j, k, m);
}
```

Outputs `-2 2 0 1`

- Precedence of `&&` greater than `||`, but short circuit evaluation occurs in `A || (B && C)`.
- So, RHS of `||`, i.e., `(B&&C)` not evaluated as LHS (i.e., `A`) is 1.