Variable Scope

Summary of Scope Rules

```
int i;
void f(int i) {
    i = 10;
}
int j;
void g(void) {
  int i = 2;
    if (i > 0) {
        int i;
        j = 16;
        i = 25;
    }
    i = 5;
}
void h(void) {
    j = 6;
    i = 15;
}
```

- `int i` and `int j` have file scope from the point of their respective declarations.
- Declarations `int i` in `f` and `g` take precedence and have block scope.
- Function `h` accesses global variables `i` and `j`
- Declaration within `if` inside function `g` has inner block scope.

One's Complement

Adding numbers

- A negative number is represented in 1's complement.
- 103: **01100111** and -97: **-01100001**
- Addition:

Number	1's complement
carry row	111111100
103	01100111
-97	10011110
result	00000101
	1
result	00000110

- Removing carry bit and adding it to result, produces 0000110.

One's Complement

Adding numbers

- 53: **00110101** and 47: **00101111**
- Addition of -53 and -47:

Number	1's complement
carry row	110000000
-53	11001010
-47	11010000
result	10011010
	1
result	10011011

- Removing carry bit and adding it to result, produces 10011011 which is a negative number.
- So, the number is -01100100 = -(64 + 32 + 4) = -100.

One's Complement

Problem with 1's complement

- 0 has two representations 11111111 and 00000000
- Requires carry to be added to the result.
- 1's complement does not work for multiplication.
  - The complements of negative numbers must be determined and product of complement must be obtained first.
  - Sign of each number must be known in advance to determine the sign of the product.

Two's Complement

- Positive numbers upto $2^{n-1} - 1$ can be represented with n bits.
- To find two complement of a negative number:
  - Obtain binary representation of magnitude
  - Find 1's complement of the resulting binary number.
  - Add 1 to the above binary number.
- To find magnitude of a two's complement:
  - Determine 1's complement of the complement number.
  - Add 1 to the above binary number.

Two's Complement

Carry is Ignored in Addition

- Notice that:

$$X + (-Y) = X + (2^B - Y) = 2^B - (Y - X) = X - Y.$$

- Eg: consider adding -53 and -47.

$$\begin{aligned} -00110101 + (-00101111) &= (11001011)_{tc} + (11010001)_{tc} \\ &= (10011100)_{tc} \end{aligned}$$

Note: carry is ignored in 2's compelement addition.

- So, magnitude is $01100011 + 1 = 01100100 = 100_{10}$.

Two's Complement

Example

Multiplication is simple in two's complement.

-12	:	11110100
8	:	00001000
Result	:	11110100000

- Drop bit 9 and beyond to get the result 10100000.
- Since the leftmost bit is 1, result is negative
- Magnitude is $01011111 + 1 = 01100000 = 32{+}64 = 96$.

Two's Complement

Why It Works?

- 2's complement of negative number $-X$: $2^B - X$.
- Let X and Y be positive, so

$$-X * Y = (2^B - X) * Y = 2^B * Y - (X * Y)$$

- $B + 1$ bits are needed for 2^B, so, $2^B * Y$ is can be represented by bits in position $B + 1$th and beyond.
- Therefore, if we drop all bits after bit position B, we are left with $-(X * Y)$.
- So, multiplying the two's complement of X by Y is same as multiplying X by Y and then taking the two's complement.

Floating Point Numbers

Real numbers

- Numbers represented in a computer are limited by word size.
- If some bits are used for fraction, then only limited real numbers can be represented.
- A compact scientific representation is: $f \times b^k$.
- Floating point numbers use this form, separating significant digits from the magnitude.
- A fractional part consisting of significant digits would need few bits.
- The fractional part is multiplied with b^k provides near approximation to a desired real number.

Floating Point Numbers

Real Numbers

- For computer representation $b = 2$, thus, the format of a floating point number is: | sign bit | 8-bit exponent | 23-bit mantissa |
- Exponent is stored in excess 127, i.e., stored exponent is 127 more than actual exponent.
- I.e., a stored value of 1 means exponent value is -126.
- The range is $\pm 2^{-127}, \pm 2^{127}$, or $\pm 10^{-38}, \pm 10^{38}$.

Floating Point Numbers

Normalized Form

- The exponent is adjusted until the left most digit in fraction is nonzero.
- MSB of mantissa is always 1 due to above adjustment, so it can be shifted left by one bit to allow 1 extra bit of precision.
- Increasing exponent: reduces the magnitude of the fractional part.
- Having leading zeros amounts to dropping of one or more digits from a fixed sized fraction.
- After a calculation if the left most digit $= 0$, the exponent is decreased to eliminate leading zeros.