Discussion and errata

CS10003 : Programming and Data Structures (Theory)

Representation of Numbers

1s Complement Representation: To convert any binary number into 1s complement, we simply need to invert the given binary number.

$$\boxed{1}\boxed{1}\boxed{0}\boxed{1} \longrightarrow \boxed{0}\boxed{0}\boxed{1}\boxed{0}$$

2s Complement Representation: To convert any binary number into 2s complement, we simply need to add 1 to its 1s complement form.

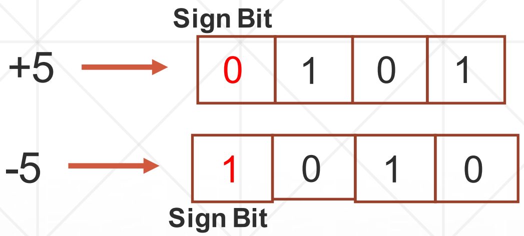
$$\boxed{1}\boxed{1}\boxed{0}\boxed{1} \xrightarrow{\text{1s Com}} \boxed{0}\boxed{0}\boxed{1}\boxed{0} \xrightarrow{+1} \boxed{0}\boxed{0}\boxed{1}\boxed{1}$$

They are generally useful in signed number representation

Signed Number Representation

1s Complement:

- Positive numbers are represented as standard binary numbers with a signed bit.
- Negative numbers are represented in 1s complement format



Sign Bit

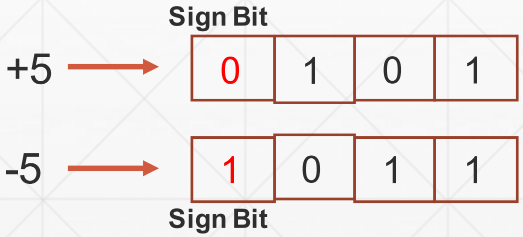
+5 → 0 1 0 1

-5 → 1 0 1 0

Sign Bit

2s Complement:

- Positive numbers are represented as standard binary numbers with a signed bit.
- Negative numbers are represented in 2s complement format



Sign Bit

+5 → 0 1 0 1

-5 → 1 0 1 1

Sign Bit

Signed Number Arithmetic

1s Complement:

Case 1:

Addition of positive and negative number when positive number has greater magnitude

Add 01110(14) and -01101(-13): 01110 + 10010 = 1 00000. We add the carry bit back to the sum. So the result is 0001.

Case 2:

Addition of positive and negative number when negative number has greater magnitude

Add -01110(-14) and 01101(13): 10001 + 01101 = 11110. We take the 1s complement of the sum, which is equivalent to -1.

Case 3:

Addition of two negative numbers

Add -01110(-14) and -01101(-13): 10001 + 10010 = 1 00011. We add the carry bit back to the sum. So the result is 00100. We take the 1s complement of the result, which is equivalent to -27.

Signed Number Arithmetic

2s Complement:

Case 1:

Addition of positive and negative number when positive number has greater magnitude

Add 01110(14) and -01101(-13): 01110 + 10011 = 1 00001. We just have to discard the carry bit from the sum.

Case 2:

Addition of positive and negative number when negative number has greater magnitude

Add -01110(-14) and 01101(13): 10010 + 01101 = 11111. We take the 2s complement of the sum, which is equivalent to -1.

Case 3:

Addition of two negative numbers

Add -01110(-14) and -01101(-13): 10010 + 10011 = 1 00101. We discard the carry bit from the sum. So the result is 00101. We take the 2s complement of the result, which is equivalent to -27.

Question

Add -18 and -21:

Question

Add -18 and -21:

- 101110 + 101011 = 1 011001.

Question

Add -18 and -21:

- 101110 + 101011 = 1 011001.
- We discard the carry bit from the sum. So the result is 011001.

Question

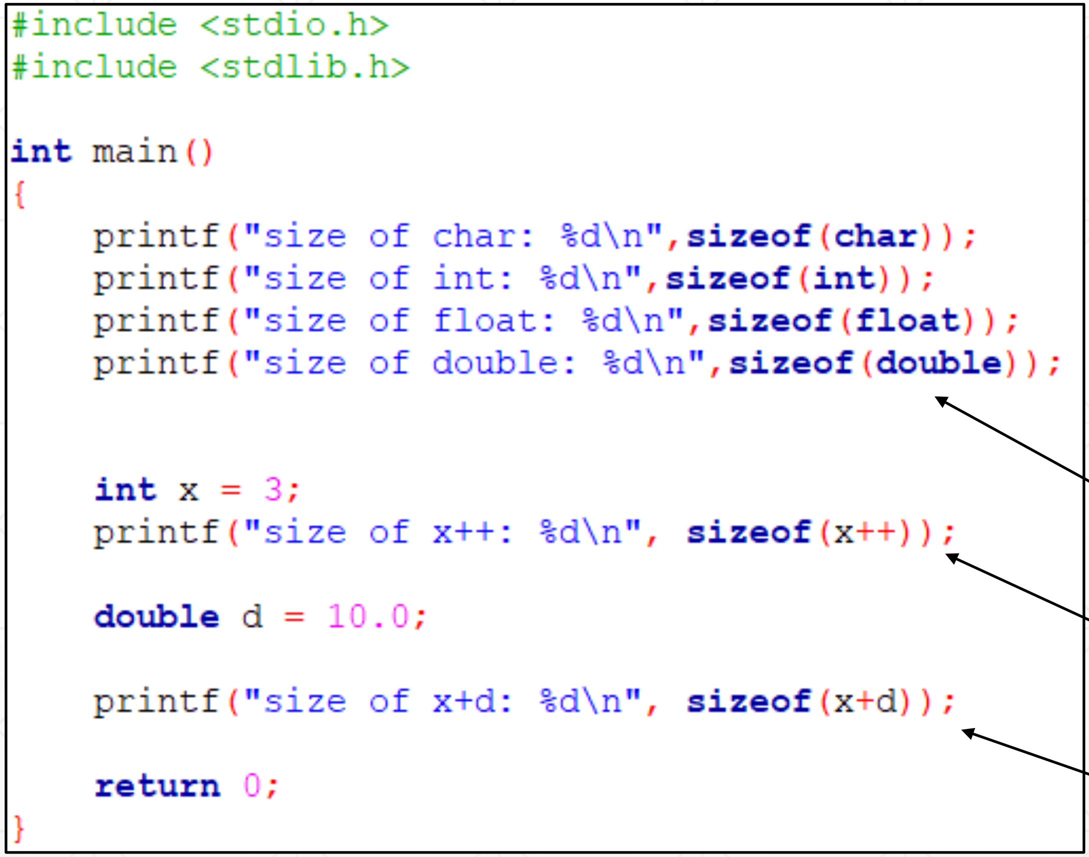
Add -18 and -21:

- 101110 + 101011 = 1 011001.
- We discard the carry bit from the sum. So the result is 011001.
- We take the 2s complement of the result, which is equivalent to -49.

The sizeof function (Lecture 2, Part-2, Slide number 5)

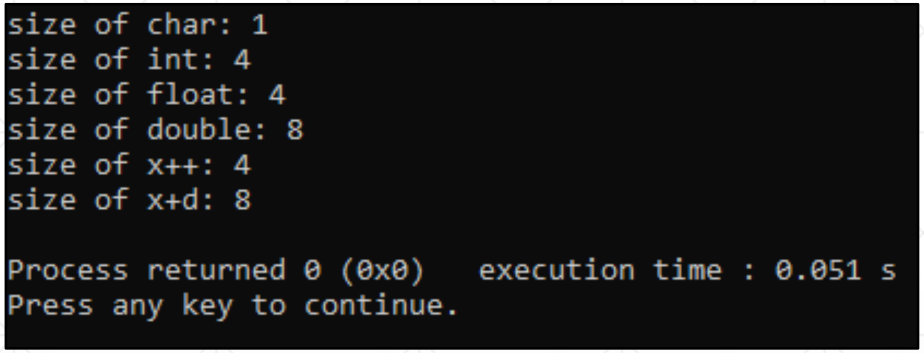
- The ***sizeof*** is **not a real function**. Instead it is considered as an **operator**.
- It is used to compute the size of its operand.
- Operand can be a data type (e.g. char, int, float) or an expression (e.g. a+b).
- ***sizeof*** operator **doesn’t need to evaluate** the expression to obtain the size as the data type of the operand doesn’t change and hence the size remains the same.

Example Code of sizeof operator in C



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("size of char: %d\n",sizeof(char));
    printf("size of int: %d\n",sizeof(int));
    printf("size of float: %d\n",sizeof(float));
    printf("size of double: %d\n",sizeof(double));


    int x = 3;
    printf("size of x++: %d\n", sizeof(x++));

    double d = 10.0;

    printf("size of x+d: %d\n", sizeof(x+d));

    return 0;
}
```

Output :



```
size of char: 1
size of int: 4
size of float: 4
size of double: 8
size of x++: 4
size of x+d: 8

Process returned 0 (0x0)   execution time : 0.051 s
Press any key to continue.
```

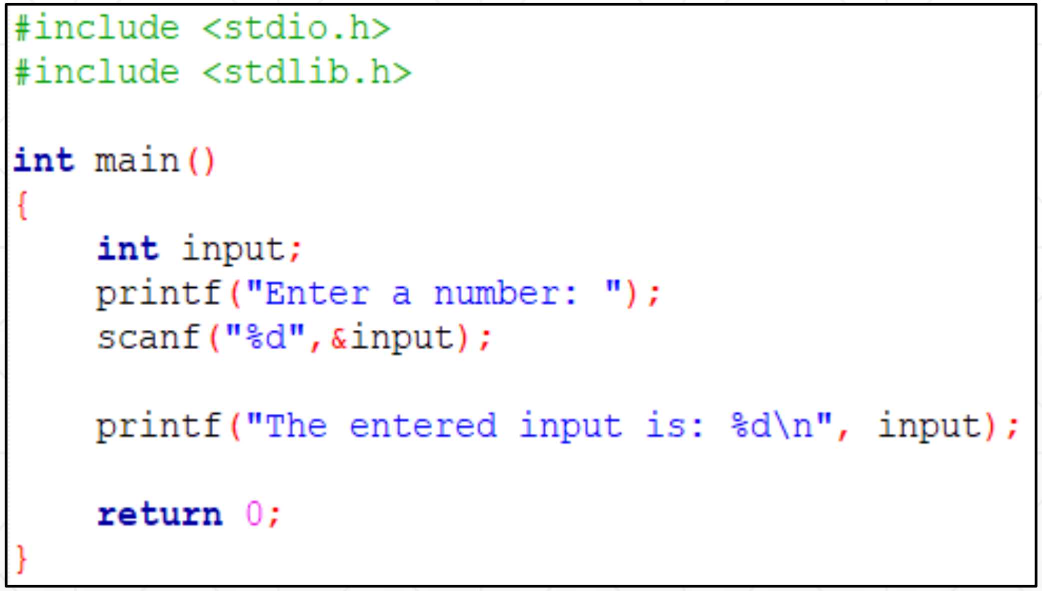
Operand is a data type

Doesn’t need to evaluate the expression

Operand is an expression

Integer Constants (Lecture 2, Part-2, Slide number 20)

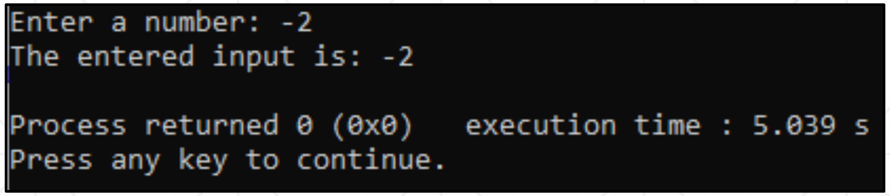
Example Code in C :



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int input;
    printf("Enter a number: ");
    scanf("%d",&input);

    printf("The entered input is: %d\n", input);

    return 0;
}
```

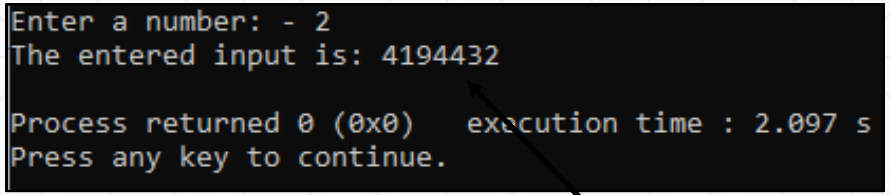
Output :

Scenario 1 :



```
Enter a number: -2
The entered input is: -2

Process returned 0 (0x0)   execution time : 5.039 s
Press any key to continue.
```

Scenario 2 :



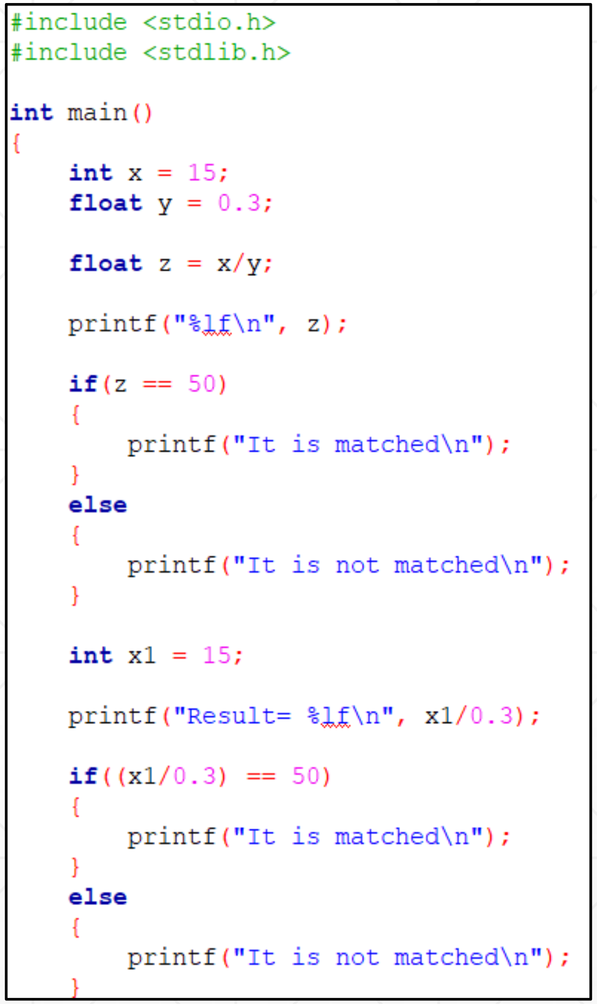
```
Enter a number: - 2
The entered input is: 4194432

Process returned 0 (0x0)   execution time : 2.097 s
Press any key to continue.
```

Garbage value is printed

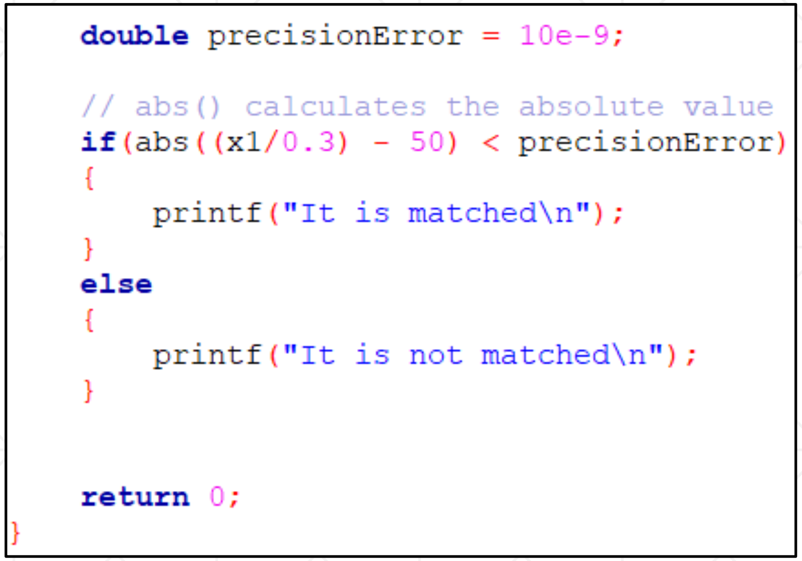
Floating point errors (Lecture 2, Part-3, Slide number 43)

- In the case of floating-point numbers, the relational operator (==) does not produce correct output, this is due to the internal **precision errors** in rounding up floating-point numbers.
- Internal rounding error in floating-point numbers.
- May vary from system to system.

Example Code of Floating point errors in C :

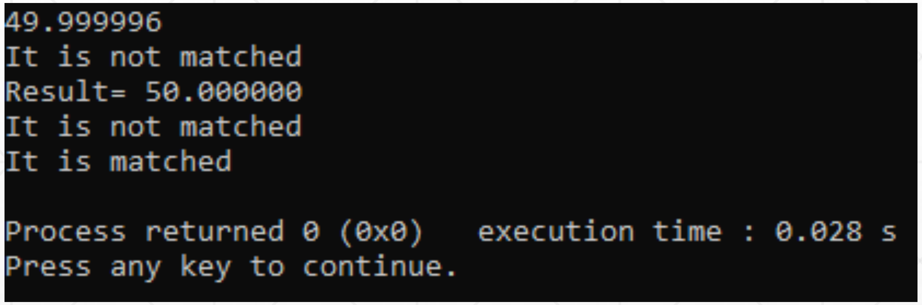


```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x = 15;
    float y = 0.3;

    float z = x/y;

    printf("%lf\n", z);

    if(z == 50)
    {
        printf("It is matched\n");
    }
    else
    {
        printf("It is not matched\n");
    }

    int x1 = 15;

    printf("Result= %lf\n", x1/0.3);

    if((x1/0.3) == 50)
    {
        printf("It is matched\n");
    }
    else
    {
        printf("It is not matched\n");
    }
```



```c
    double precisionError = 10e-9;

    // abs() calculates the absolute value
    if(abs((x1/0.3) - 50) < precisionError)
    {
        printf("It is matched\n");
    }
    else
    {
        printf("It is not matched\n");
    }


    return 0;
}
```

Output :



```
49.999996
It is not matched
Result= 50.000000
It is not matched
It is matched

Process returned 0 (0x0)   execution time : 0.028 s
Press any key to continue.
```

Floating point errors (Lecture 2, Part-3, Slide number 43)

- In the first scenario, initial value of z is 49.999996. z is not correctly rounded up to 50 due to an internal error in rounding up, a very small error but makes a huge difference when we are comparing the numbers.

- In the second scenario, the same error occurs due to comparison between mismatched data types.

- If we need to compare two floating-point numbers then rather than using “==” operator we will find the absolute difference between the numbers and compare in against a very small number (10^{-9}) as shown in the third scenario.

- This scenario may vary from system to system.