Advanced Programming

Lab 02

CONTENTS

1. Objectives

- Master Fundamental Data types
- Master Arithmetic Operators and Assignment Operators
- Master Keyboard Input and Terminal Output

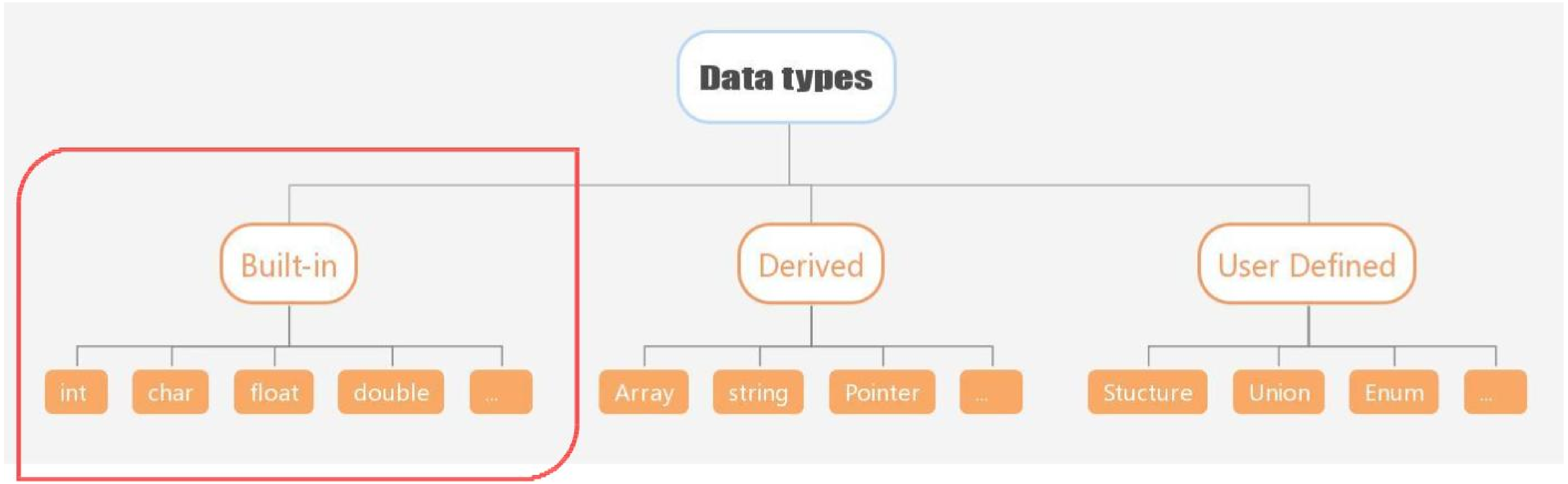
2 Knowledge Points

2.1 Fundamental Data Types

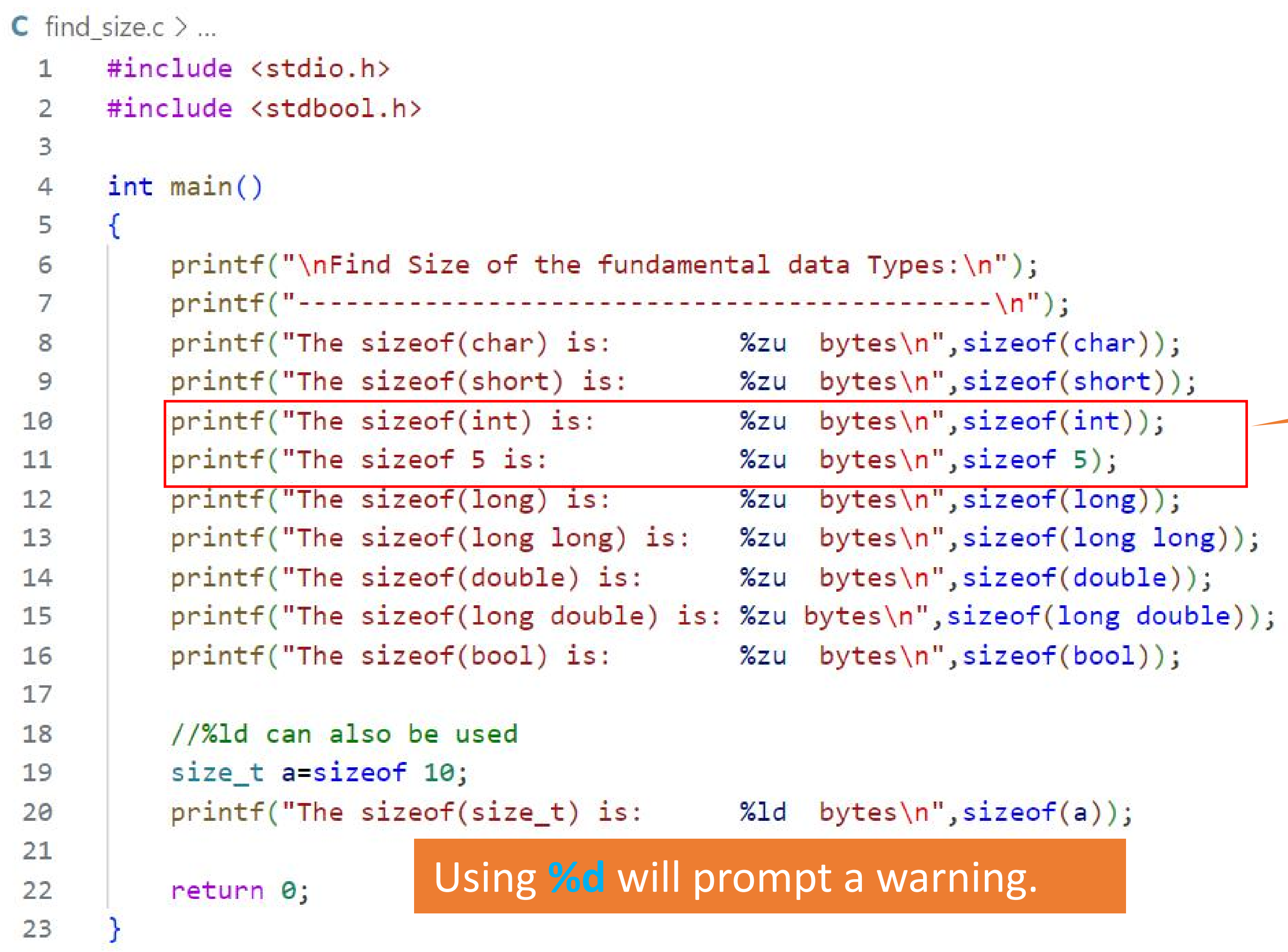
2.2 Arithmetic Operators and Assignment Operators

2.3 Input and Output

2.1 Data types



sizeof operator returns the size, in bytes, of a type or a variable.



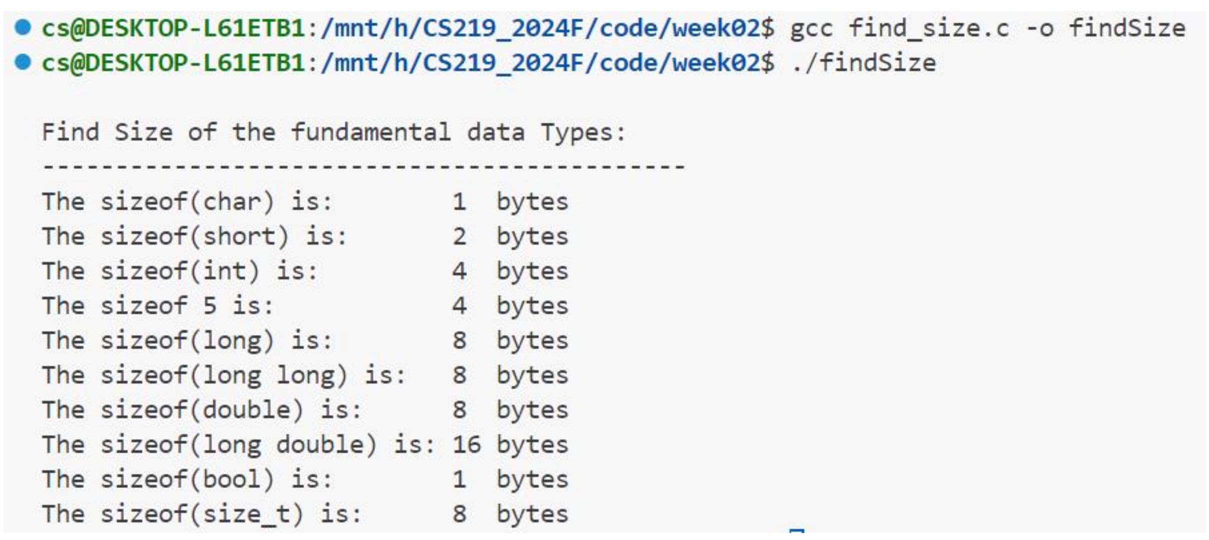
C find_size.c > ...

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

int main()
{
    printf("\nFind Size of the fundamental data Types:\n");
    printf("-------------------------------------------\n");
    printf("The sizeof(char) is:        %zu  bytes\n",sizeof(char));
    printf("The sizeof(short) is:       %zu  bytes\n",sizeof(short));
    printf("The sizeof(int) is:         %zu  bytes\n",sizeof(int));
    printf("The sizeof 5 is:            %zu  bytes\n",sizeof 5);
    printf("The sizeof(long) is:        %zu  bytes\n",sizeof(long));
    printf("The sizeof(long long) is:   %zu  bytes\n",sizeof(long long));
    printf("The sizeof(double) is:      %zu  bytes\n",sizeof(double));
    printf("The sizeof(long double) is: %zu bytes\n",sizeof(long double));
    printf("The sizeof(bool) is:        %zu  bytes\n",sizeof(bool));

    //%ld can also be used
    size_t a=sizeof 10;
    printf("The sizeof(size_t) is:      %ld  bytes\n",sizeof(a));

    return 0;
}
```

using the **sizeof** operator with a type name, enclose the name in parentheses; while using the operator with the name of the variable, parentheses are optional.

Using %d will prompt a warning.

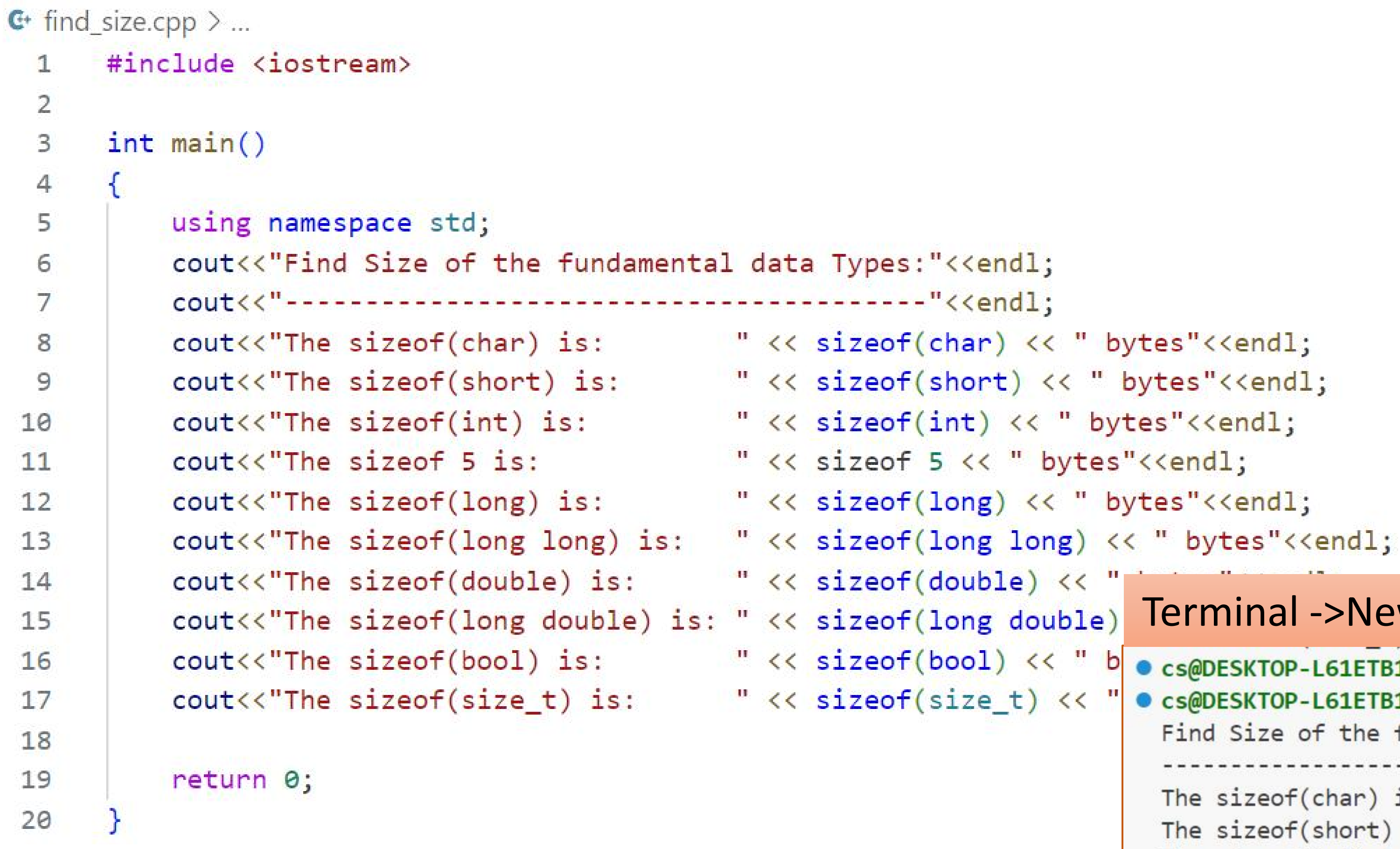
Terminal ->New Terminal



```
● cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ gcc find_size.c -o findSize
● cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ ./findSize

 Find Size of the fundamental data Types:
 -------------------------------------------
 The sizeof(char) is:        1  bytes
 The sizeof(short) is:       2  bytes
 The sizeof(int) is:         4  bytes
 The sizeof 5 is:            4  bytes
 The sizeof(long) is:        8  bytes
 The sizeof(long long) is:   8  bytes
 The sizeof(double) is:      8  bytes
 The sizeof(long double) is: 16 bytes
 The sizeof(bool) is:        1  bytes
 The sizeof(size_t) is:      8  bytes
```

Reference: https://en.cppreference.com/w/c/types/size_t

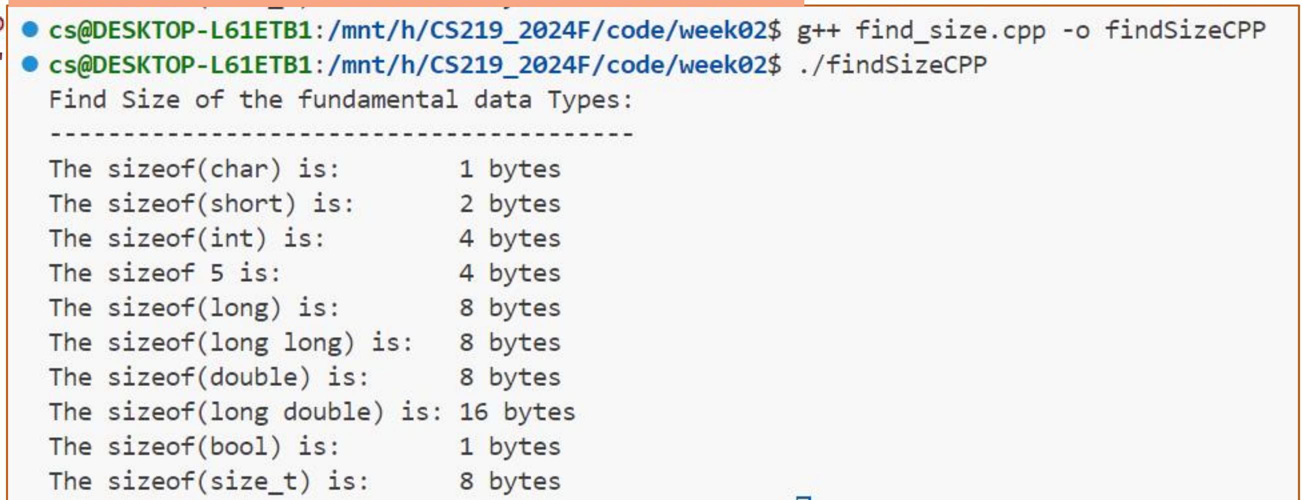
C++ program are similar:



find_size.cpp > ...

```cpp
#include <iostream>

int main()
{
    using namespace std;
    cout<<"Find Size of the fundamental data Types:"<<endl;
    cout<<"----------------------------------------"<<endl;
    cout<<"The sizeof(char) is:        " << sizeof(char) << " bytes"<<endl;
    cout<<"The sizeof(short) is:       " << sizeof(short) << " bytes"<<endl;
    cout<<"The sizeof(int) is:         " << sizeof(int) << " bytes"<<endl;
    cout<<"The sizeof 5 is:            " << sizeof 5 << " bytes"<<endl;
    cout<<"The sizeof(long) is:        " << sizeof(long) << " bytes"<<endl;
    cout<<"The sizeof(long long) is:   " << sizeof(long long) << " bytes"<<endl;
    cout<<"The sizeof(double) is:      " << sizeof(double) << "
    cout<<"The sizeof(long double) is: " << sizeof(long double)
    cout<<"The sizeof(bool) is:        " << sizeof(bool) << " b
    cout<<"The sizeof(size_t) is:      " << sizeof(size_t) << "

    return 0;
}
```

Terminal ->New Terminal



```
● cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ g++ find_size.cpp -o findSizeCPP
● cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ ./findSizeCPP
  Find Size of the fundamental data Types:
  ----------------------------------------
  The sizeof(char) is:        1 bytes
  The sizeof(short) is:       2 bytes
  The sizeof(int) is:         4 bytes
  The sizeof 5 is:            4 bytes
  The sizeof(long) is:        8 bytes
  The sizeof(long long) is:   8 bytes
  The sizeof(double) is:      8 bytes
  The sizeof(long double) is: 16 bytes
  The sizeof(bool) is:        1 bytes
  The sizeof(size_t) is:      8 bytes
```

The value range of an integer

```cpp
#include <iostream>
#include <climits>
using namespace std;

int main()
{
    int n_int = INT_MAX; // initialize n_int to max int value
    short n_short = SHRT_MAX; // symbols defined in climits file
    long n_long = LONG_MAX;
    long long n_llong = LLONG_MAX;
    // sizeof operator yields size of type or of variable
    cout << "int is " << sizeof (int) << " bytes." << endl;
    cout << "short is " << sizeof n_short << " bytes." << endl;
    cout << "long is " << sizeof n_long << " bytes." << endl;
    cout << "long long is " << sizeof n_llong << " bytes." << endl;
    cout << endl;
    cout << "Maximum values:" << endl;
    cout << "int: " << n_int << endl;
    cout << "short: " << n_short << endl;
    cout << "long: " << n_long << endl;
    cout << "long long: " << n_llong << endl << endl;
    cout << "Minimum int value = " << INT_MIN << endl;
    cout << "Bits per byte = " << CHAR_BIT << endl;
    return 0;
}
```

number of bits in a byte

Reference:
https://en.cppreference.com/w/cpp/types/climits

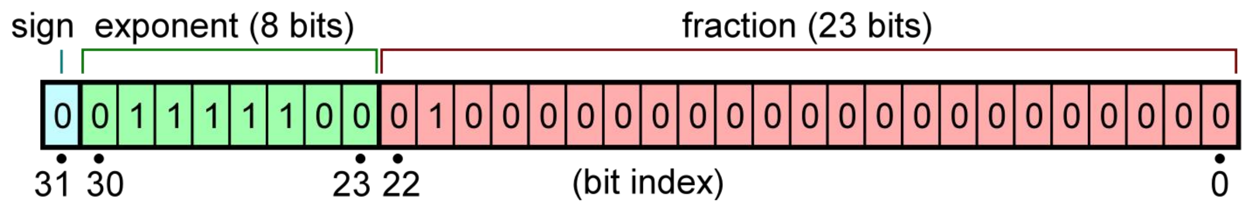
SCHAR_MAX SHRT_MAX INT_MAX LONG_MAX LLONG_MAX (C99)	maximum value of signed char, short, int, long and long long respectively (macro constant)

```
● cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ g++ limits.cpp -o limits
● cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ ./limits
  int is 4 bytes.
  short is 2 bytes.
  long is 8 bytes.
  long long is 8 bytes.

  Maximum values:
  int: 2147483647
  short: 32767
  long: 9223372036854775807
  long long: 9223372036854775807

  Minimum int value = -2147483648
  Bits per byte = 8
```

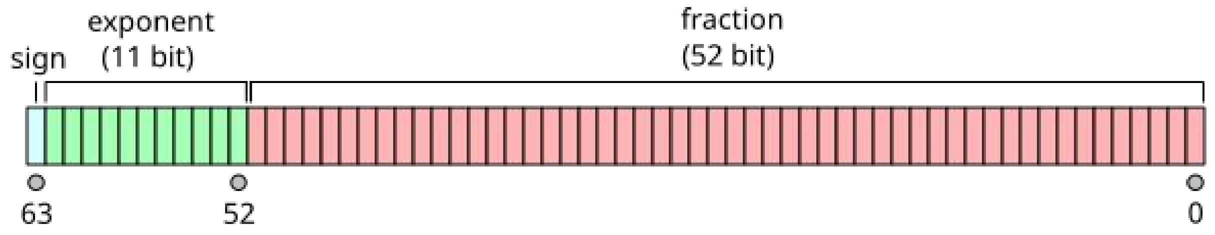
Floating point precision

Floating-point type also has a range of values. Besides that, it has a significant figures. Normally, the system guarantees the 6 significant figures of type float variable, and 15 significant figure of the type double variable. Floating-point numbers have precision limitations when they are evaluated.

Single-precision floating-point format

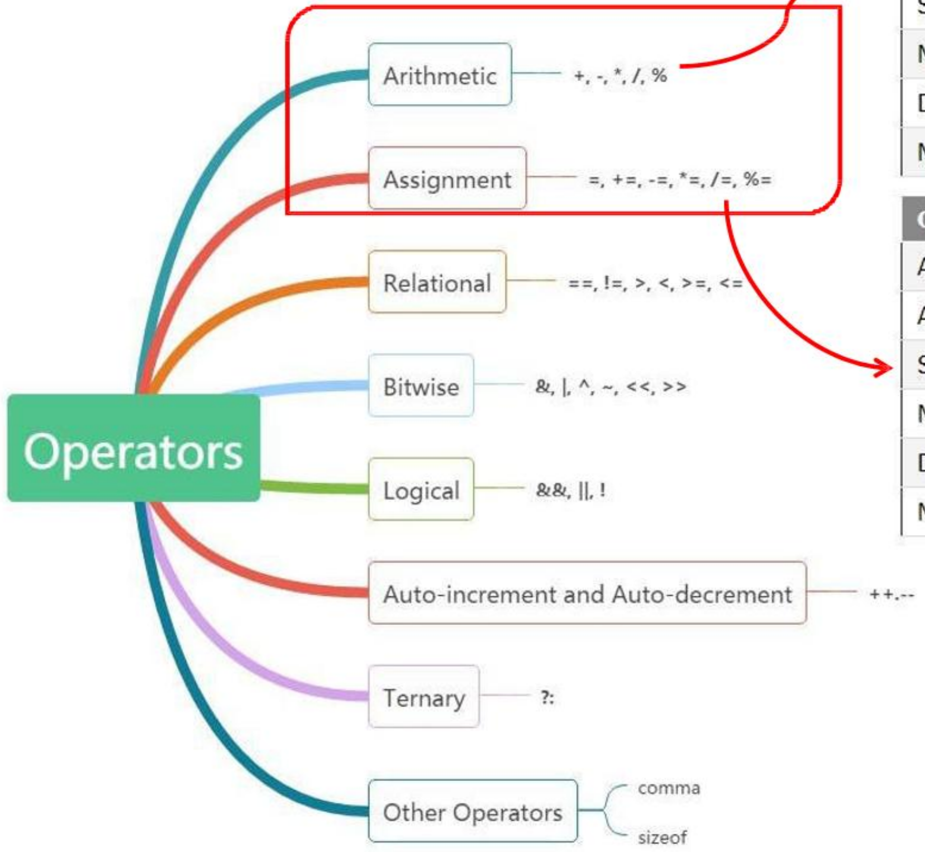


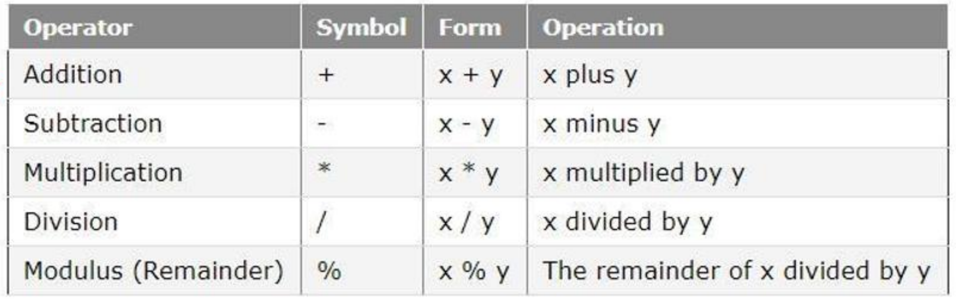
Double-precision floating-point format



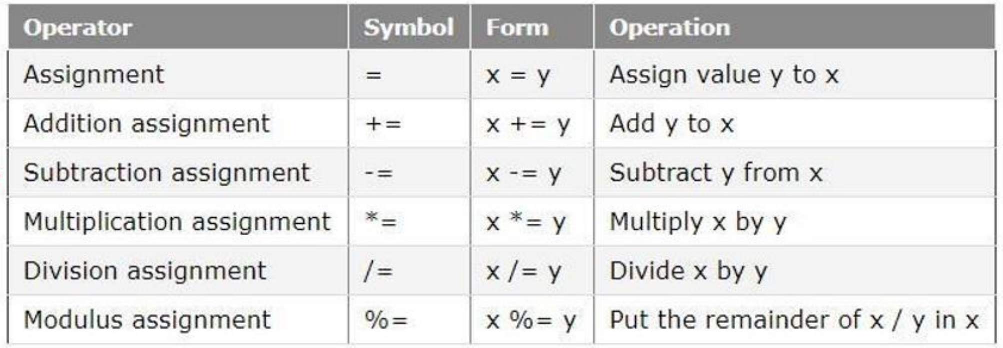
Reference :
https://en.wikipedia.org/wiki/Double-precision_floating-point_format
https://en.wikipedia.org/wiki/Single-precision_floating-point_format

2.2 Arithmetic Operators



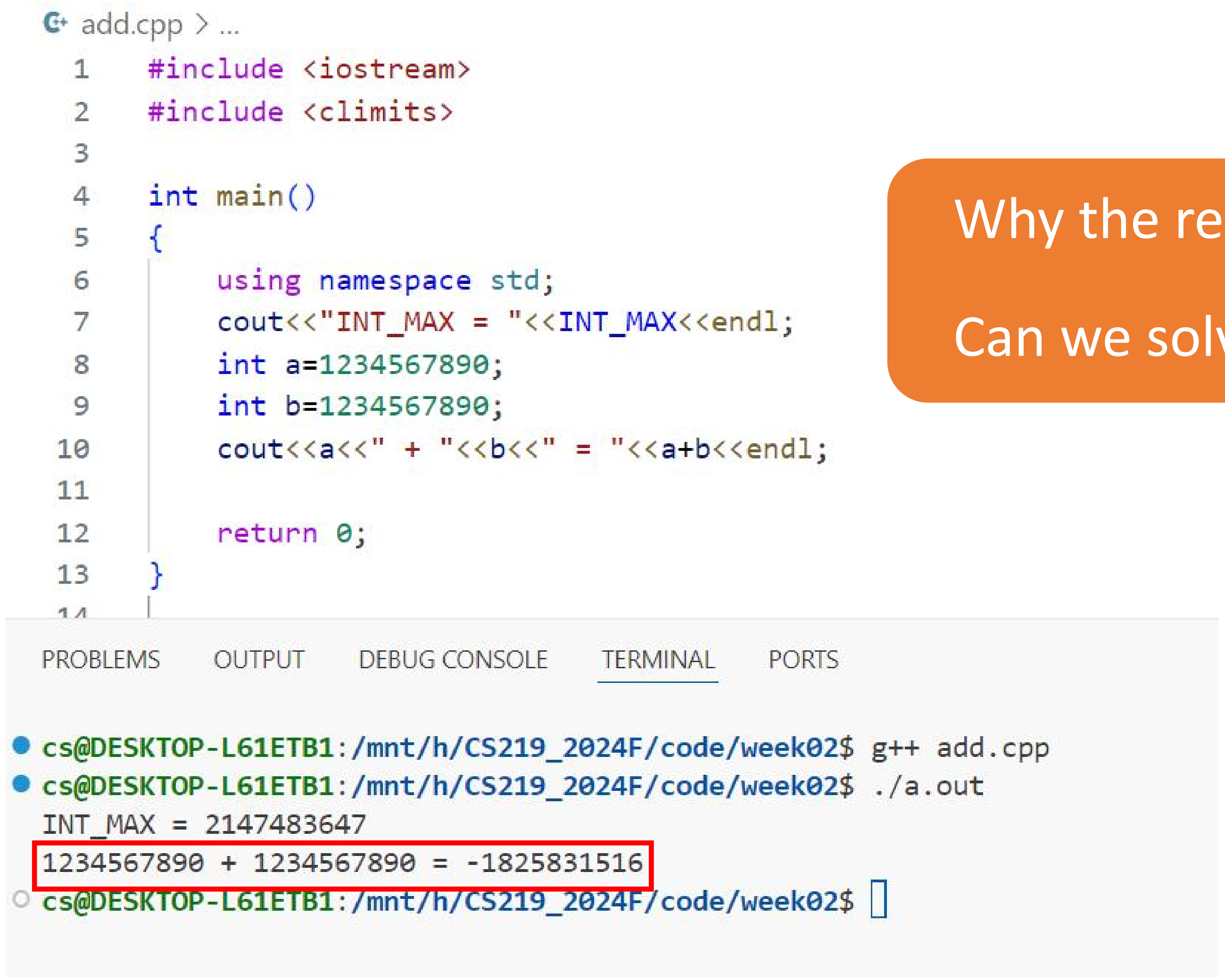


Operator	Symbol	Form	Operation
Addition	+	x + y	x plus y
Subtraction	-	x - y	x minus y
Multiplication	*	x * y	x multiplied by y
Division	/	x / y	x divided by y
Modulus (Remainder)	%	x % y	The remainder of x divided by y



Operator	Symbol	Form	Operation
Assignment	=	x = y	Assign value y to x
Addition assignment	+=	x += y	Add y to x
Subtraction assignment	-=	x -= y	Subtract y from x
Multiplication assignment	*=	x *= y	Multiply x by y
Division assignment	/=	x /= y	Divide x by y
Modulus assignment	%=	x %= y	Put the remainder of x / y in x

Example Program 1 of Arithmetic Operators



```cpp
#include <iostream>
#include <climits>

int main()
{
    using namespace std;
    cout<<"INT_MAX = "<<INT_MAX<<endl;
    int a=1234567890;
    int b=1234567890;
    cout<<a<<" + "<<b<<" = "<<a+b<<endl;

    return 0;
}
```

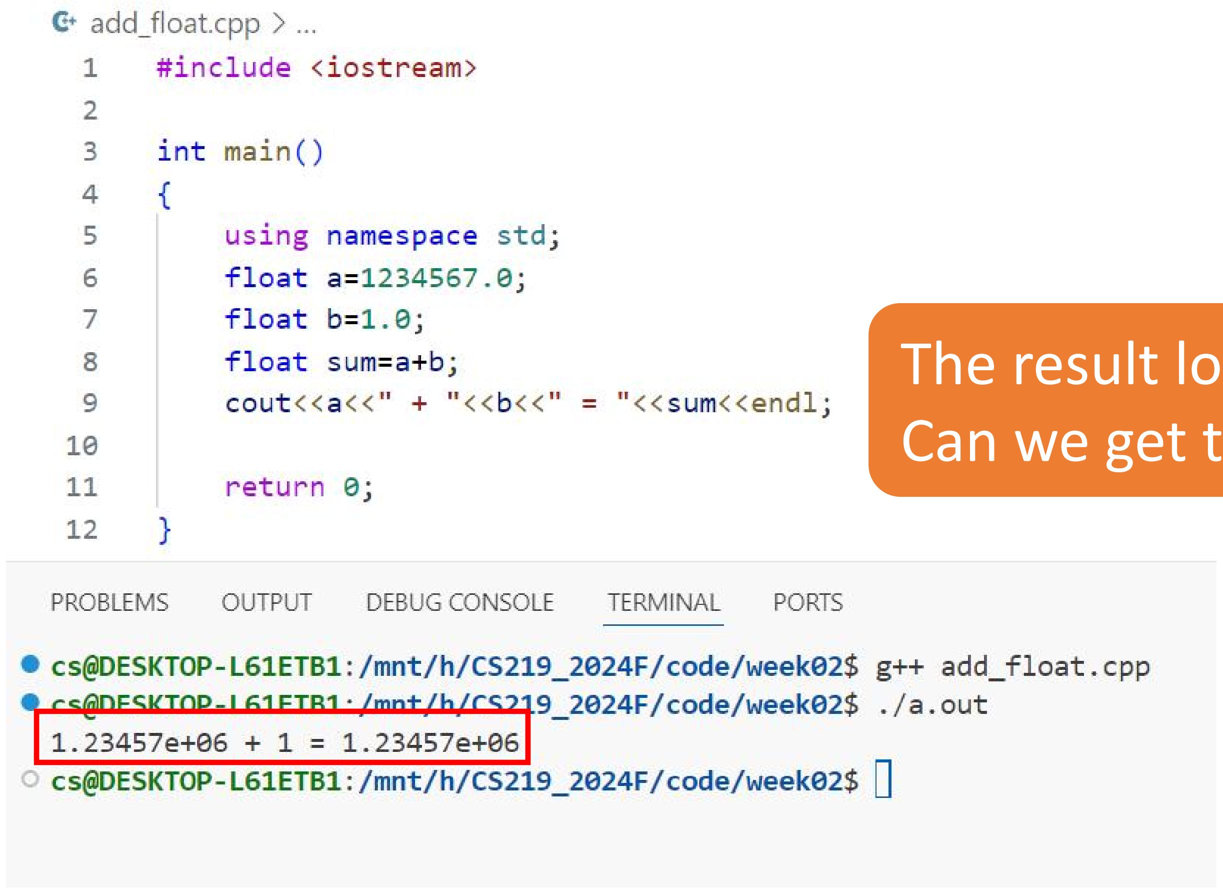
```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ g++ add.cpp
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ ./a.out
INT_MAX = 2147483647
1234567890 + 1234567890 = -1825831516
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$
```

Why the result is negative?

Can we solve the this problem?

Tips: use unsigned int or long type

Example Program 2 of Arithmetic Operators


add_float.cpp > ...

```cpp
#include <iostream>

int main()
{
    using namespace std;
    float a=1234567.0;
    float b=1.0;
    float sum=a+b;
    cout<<a<<" + "<<b<<" = "<<sum<<endl;

    return 0;
}
```

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ g++ add_float.cpp
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ ./a.out
1.23457e+06 + 1 = 1.23457e+06
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$
```

The result looks like no addition performed. Why?
Can we get the right result?

Tips: use fixed-point notation

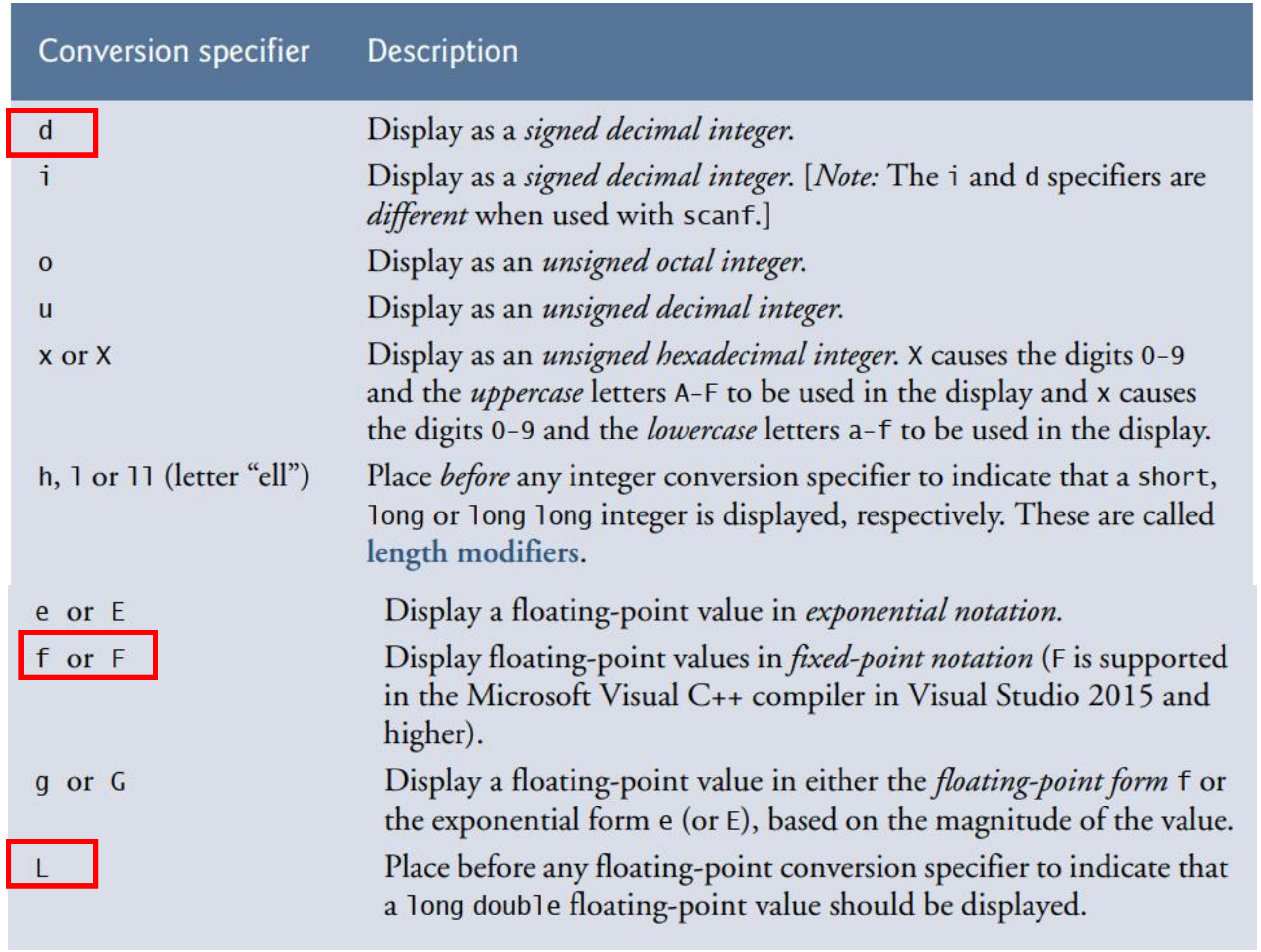
https://en.cppreference.com/w/cpp/io/manip/setprecision

2.3 Keyboard input and terminal output

2.3.1.Formatting output with *printf*

printf (*format-control-string, other-arguments*)

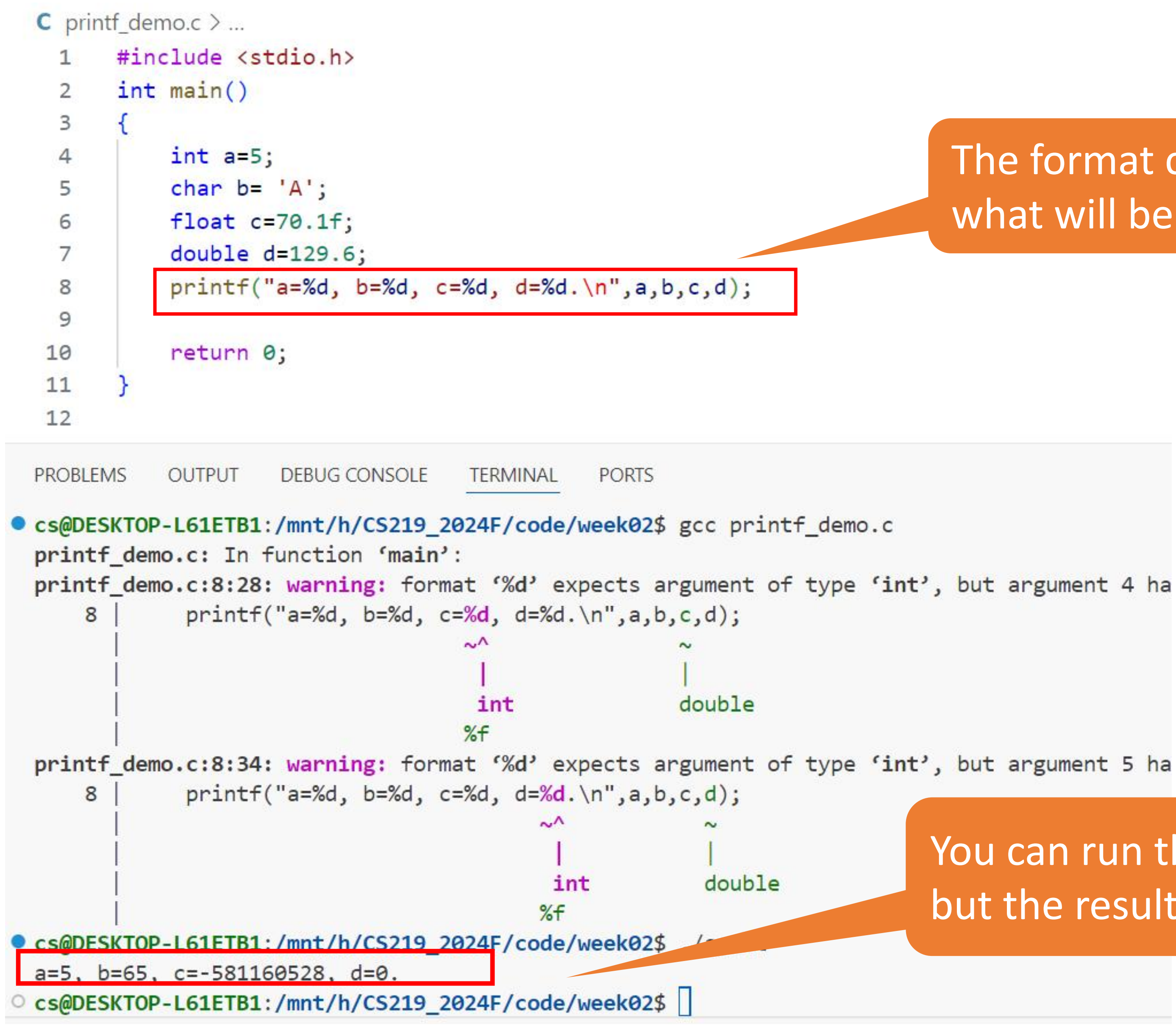
format-control-string describes the output format, which consists of conversion specifiers, field widths, precisions and literal characters with percent sign(%).



Conversion specifier	Description
`d`	Display as a *signed decimal integer.*
`i`	Display as a *signed decimal integer.* [*Note:* The `i` and `d` specifiers are *different* when used with `scanf`.]
`o`	Display as an *unsigned octal integer.*
`u`	Display as an *unsigned decimal integer.*
`x` or `X`	Display as an *unsigned hexadecimal integer.* `X` causes the digits `0-9` and the *uppercase* letters `A-F` to be used in the display and `x` causes the digits `0-9` and the *lowercase* letters `a-f` to be used in the display.
`h`, `l` or `ll` (letter "ell")	Place *before* any integer conversion specifier to indicate that a `short`, `long` or `long long` integer is displayed, respectively. These are called **length modifiers**.
`e` or `E`	Display a floating-point value in *exponential notation.*
`f` or `F`	Display floating-point values in *fixed-point notation* (`F` is supported in the Microsoft Visual C++ compiler in Visual Studio 2015 and higher).
`g` or `G`	Display a floating-point value in either the *floating-point form* `f` or the exponential form `e` (or `E`), based on the magnitude of the value.
`L`	Place before any floating-point conversion specifier to indicate that a `long double` floating-point value should be displayed.

Reference: https://www.w3cschool.cn/c/c-function-printf.html

Example Program of printf



```c
1   #include <stdio.h>
2   int main()
3   {
4       int a=5;
5       char b= 'A';
6       float c=70.1f;
7       double d=129.6;
8       printf("a=%d, b=%d, c=%d, d=%d.\n",a,b,c,d);
9
10      return 0;
11  }
12
```

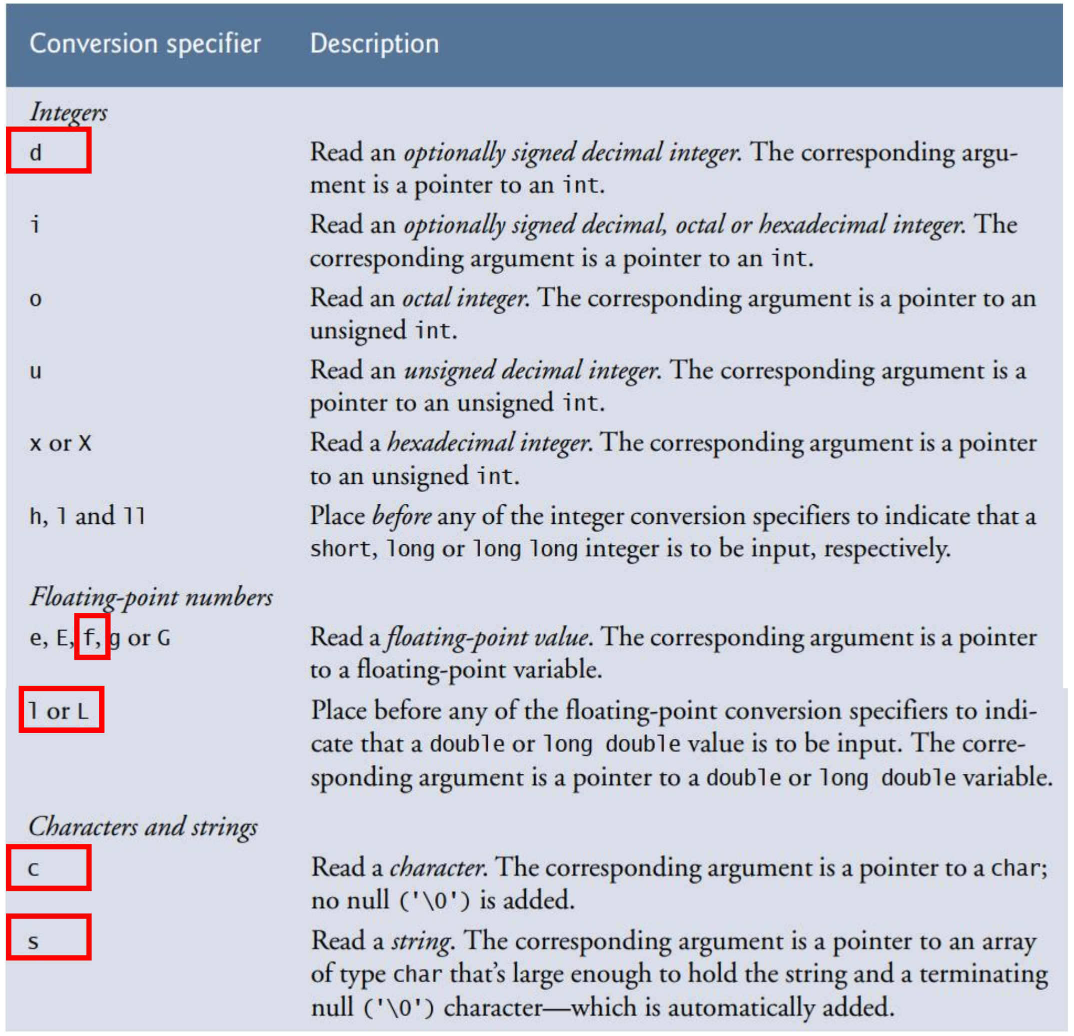
```
● cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ gcc printf_demo.c
  printf_demo.c: In function ‘main’:
  printf_demo.c:8:28: warning: format ‘%d’ expects argument of type ‘int’, but argument 4 ha
      8 |     printf("a=%d, b=%d, c=%d, d=%d.\n",a,b,c,d);
        |                         ~^           ~
        |                          |           |
        |                          int         double
        |                         %f
  printf_demo.c:8:34: warning: format ‘%d’ expects argument of type ‘int’, but argument 5 ha
      8 |     printf("a=%d, b=%d, c=%d, d=%d.\n",a,b,c,d);
        |                               ~^           ~
        |                                |           |
        |                                int         double
        |                               %f
● cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ /
  a=5, b=65, c=-581160528, d=0.
○ cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ 
```

The format control strings don’t match the types of variables, what will be the output?

You can run the program ignoring the warnings, but the result may not be correct.

2.3.2. Reading Formatted input with *scanf*

format-control-string describes the formats of input, ***other-arguments*** are **pointers** to variables in which the input will be stored.



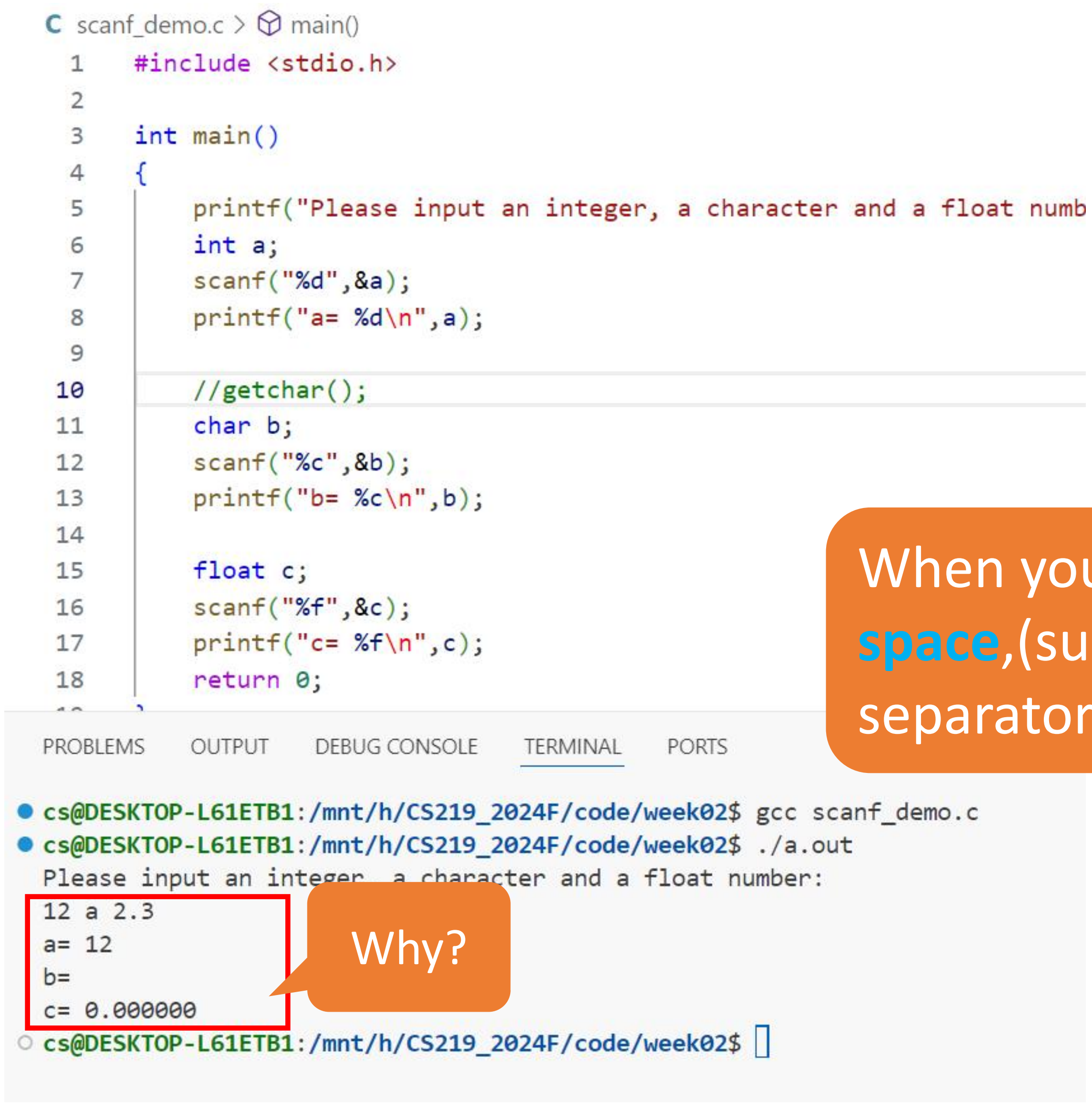
Conversion specifier	Description
Integers	
d	Read an *optionally signed decimal integer*. The corresponding argument is a pointer to an int.
i	Read an *optionally signed decimal, octal or hexadecimal integer*. The corresponding argument is a pointer to an int.
o	Read an *octal integer*. The corresponding argument is a pointer to an unsigned int.
u	Read an *unsigned decimal integer*. The corresponding argument is a pointer to an unsigned int.
x or X	Read a *hexadecimal integer*. The corresponding argument is a pointer to an unsigned int.
h, l and ll	Place *before* any of the integer conversion specifiers to indicate that a short, long or long long integer is to be input, respectively.
Floating-point numbers	
e, E, f, g or G	Read a *floating-point value*. The corresponding argument is a pointer to a floating-point variable.
l or L	Place before any of the floating-point conversion specifiers to indicate that a double or long double value is to be input. The corresponding argument is a pointer to a double or long double variable.
Characters and strings	
c	Read a *character*. The corresponding argument is a pointer to a char; no null ('\0') is added.
s	Read a *string*. The corresponding argument is a pointer to an array of type char that's large enough to hold the string and a terminating null ('\0') character—which is automatically added.

scanf (*format-control-string, other-arguments*)

Note: When inputting data, prompt the user for one data item or a few data items at a time. Avoid asking the user to enter many data items in response to a single prompt.

Reference: https://www.w3cschool.cn/c/c-function-scanf.html

Example Program of scanf

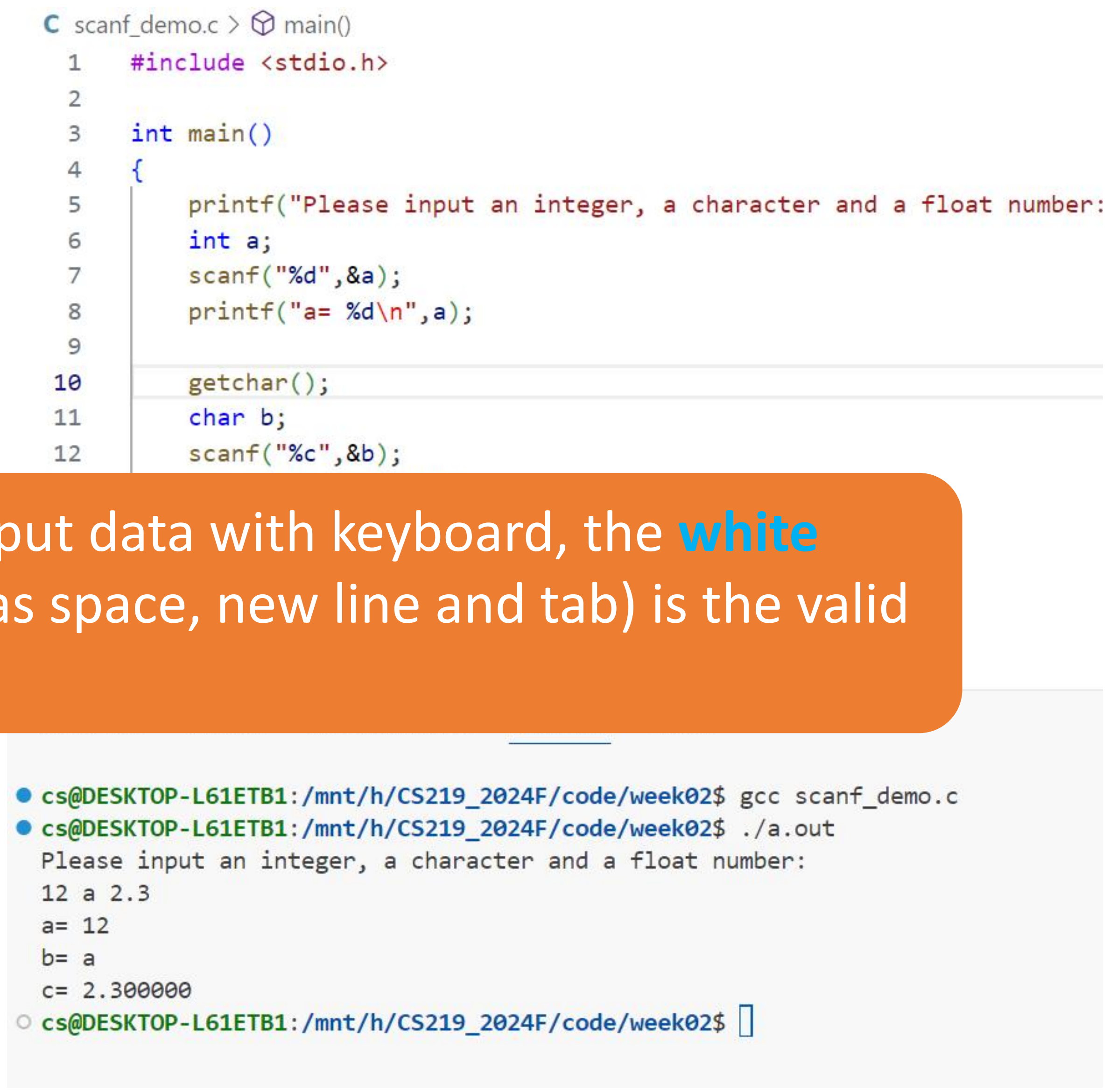


scanf_demo.c > main()

```c
#include <stdio.h>

int main()
{
    printf("Please input an integer, a character and a float numb
    int a;
    scanf("%d",&a);
    printf("a= %d\n",a);

    //getchar();
    char b;
    scanf("%c",&b);
    printf("b= %c\n",b);

    float c;
    scanf("%f",&c);
    printf("c= %f\n",c);
    return 0;
```

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ gcc scanf_demo.c
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ ./a.out
Please input an integer, a character and a float number:
12 a 2.3
a= 12
b=
c= 0.000000
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$
```

Why?



scanf_demo.c > main()

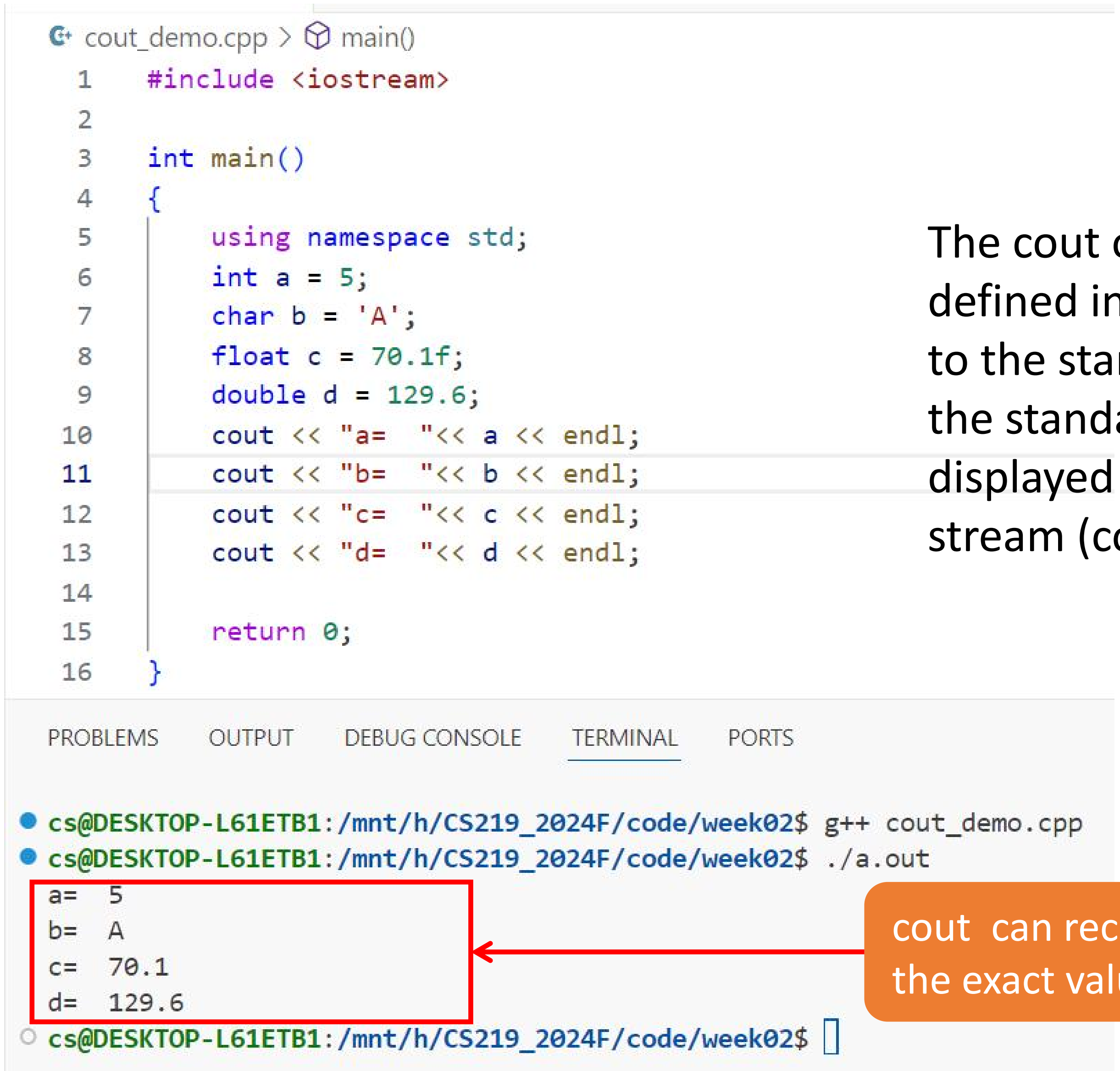
```c
#include <stdio.h>

int main()
{
    printf("Please input an integer, a character and a float number:
    int a;
    scanf("%d",&a);
    printf("a= %d\n",a);

    getchar();
    char b;
    scanf("%c",&b);
```

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ gcc scanf_demo.c
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ ./a.out
Please input an integer, a character and a float number:
12 a 2.3
a= 12
b= a
c= 2.300000
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$
```

When you input data with keyboard, the **white space**,(such as space, new line and tab) is the valid separator.

2.3.3. cout

cout << variable1(expression1) [<< variable2 << variable n];



```
cout_demo.cpp > main()
#include <iostream>

int main()
{
    using namespace std;
    int a = 5;
    char b = 'A';
    float c = 70.1f;
    double d = 129.6;
    cout << "a=  "<< a << endl;
    cout << "b=  "<< b << endl;
    cout << "c=  "<< c << endl;
    cout << "d=  "<< d << endl;

    return 0;
}
```

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ g++ cout_demo.cpp
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ ./a.out
a=  5
b=  A
c=  70.1
d=  129.6
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$
```

The cout object in C++ is an object of class iostream. It is defined in iostream header file. It is used to display the output to the standard output device i.e. monitor. It is associated with the standard C output stream stdout. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).

Floating-point types are displayed with a total of six digits, except that trailing zeros aren't displayed. In particular, E notation is used if the exponent is 6 or larger or -5 or smaller.

cout can recognizes the type of the variable and print the exact value of the variable.

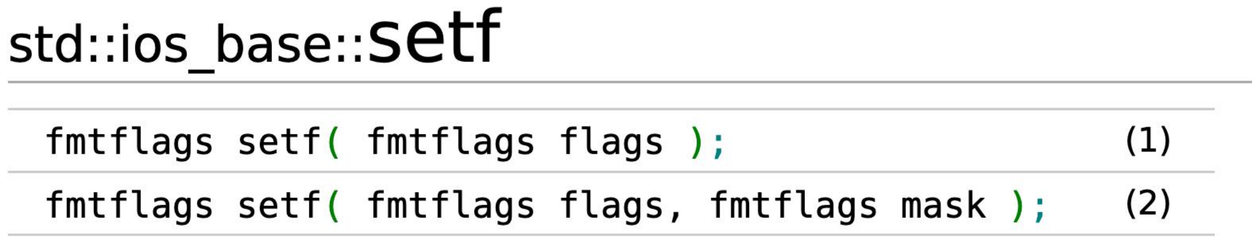
C++ provides two methods to control the **output formats**

- ***Using member functions of ios class***
- ***Using iomanip manipulators***

Using member functions of ios class

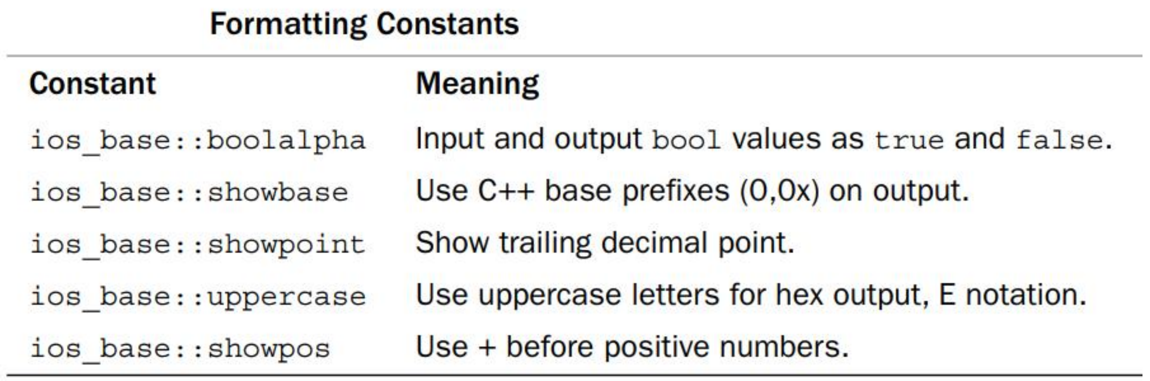
cout.setf():

The setf() function has two prototypes:



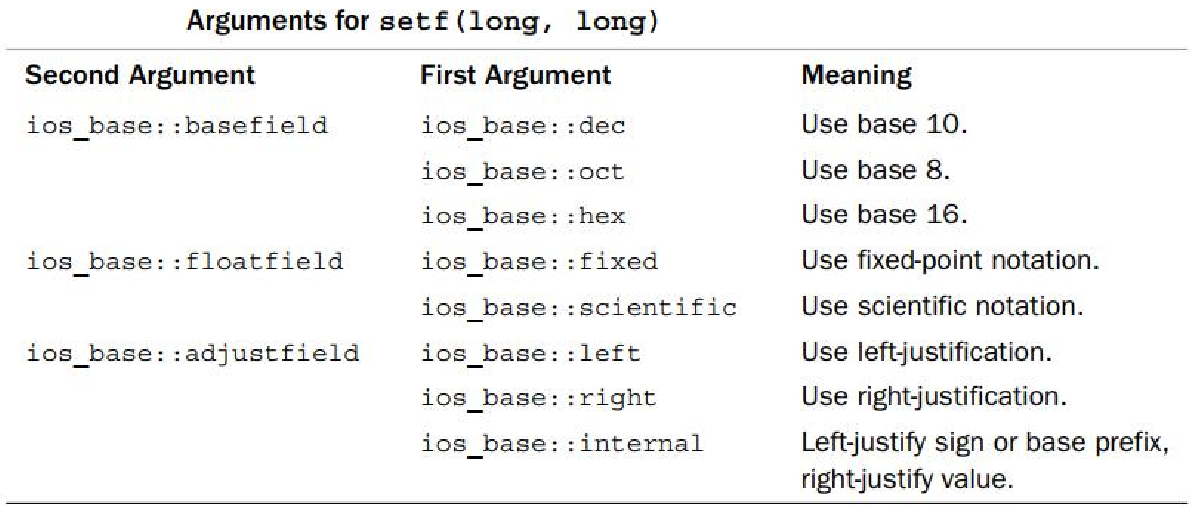
std::ios_base::setf

```
fmtflags setf( fmtflags flags );                  (1)
fmtflags setf( fmtflags flags, fmtflags mask );   (2)
```



Formatting Constants

Constant	Meaning
`ios_base::boolalpha`	Input and output `bool` values as `true` and `false`.
`ios_base::showbase`	Use C++ base prefixes (0,0x) on output.
`ios_base::showpoint`	Show trailing decimal point.
`ios_base::uppercase`	Use uppercase letters for hex output, E notation.
`ios_base::showpos`	Use + before positive numbers.



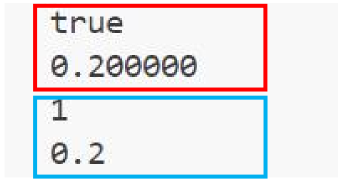
Arguments for `setf(long, long)`

Second Argument	First Argument	Meaning
`ios_base::basefield`	`ios_base::dec`	Use base 10.
	`ios_base::oct`	Use base 8.
	`ios_base::hex`	Use base 16.
`ios_base::floatfield`	`ios_base::fixed`	Use fixed-point notation.
	`ios_base::scientific`	Use scientific notation.
`ios_base::adjustfield`	`ios_base::left`	Use left-justification.
	`ios_base::right`	Use right-justification.
	`ios_base::internal`	Left-justify sign or base prefix, right-justify value.

https://cplusplus.com/reference/ios/ios_base/setf/

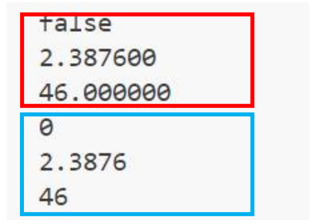
The effect of calling *setf()* can be undone with *unsef()*

```
#include <iostream>
using namespace std;

int main()
{
   bool flag = true;
   float f = 0.20f;

   cout.setf(ios::showpoint);
   cout.setf(ios::boolalpha);
   cout << flag << endl;
   cout << f << endl;

   cout.unsetf(ios::boolalpha);
   cout.unsetf(ios::showpoint);
   cout << flag << endl;
   cout << f << endl;

   return 0;
}
```



```
true
0.200000
1
0.2
```

```
#include <iostream>
using namespace std;

int main()
{
   bool flag = false;
   double a = 2.3876;
   double b = 0.46e2;

   cout << boolalpha << flag << endl;
   cout << fixed << a << endl;
   cout << b << endl;

   cout << noboolalpha << flag << endl;
   cout.unsetf(ios::fixed);
   cout << a << endl;
   cout << b << endl;

   return 0;
}
```

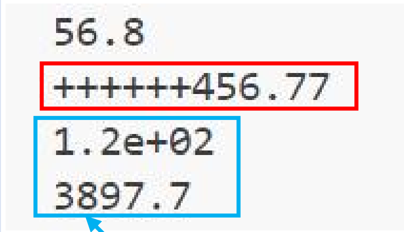


```
false
2.387600
46.000000
0
2.3876
46
```

- ***Using member functions of ios class***

2. cout.width(len) //set the field width
3. cout.fill(ch) // fill character to be used with justified field
4. cout.precision(p) // set the precision of floating-point numbers

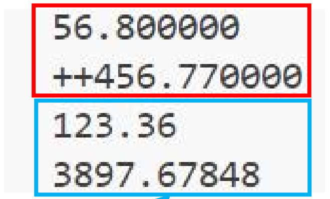
```
#include <iostream>
using namespace std;

int main()
{
   cout << 56.8 << endl;
   cout.width(12);
   cout.fill('+');
   cout << 456.77 << endl;

   cout.precision(2);
   cout << 123.356 << endl;
   cout.precision(5);
   cout << 3897.678485 << endl;

   return 0;
}
```



```
56.8
++++++456.77
1.2e+02
3897.7
```

significant digits

```
#include <iostream>
using namespace std;

int main()
{
   cout.setf(ios_base::fixed, ios_base::floatfield);
   cout << 56.8 << endl;
   cout.width(12);
   cout.fill('+');
   cout << 456.77 << endl;

   cout.precision(2);
   cout << 123.356 << endl;
   cout.precision(5);
   cout << 3897.678485 << endl;

   return 0;
}
```

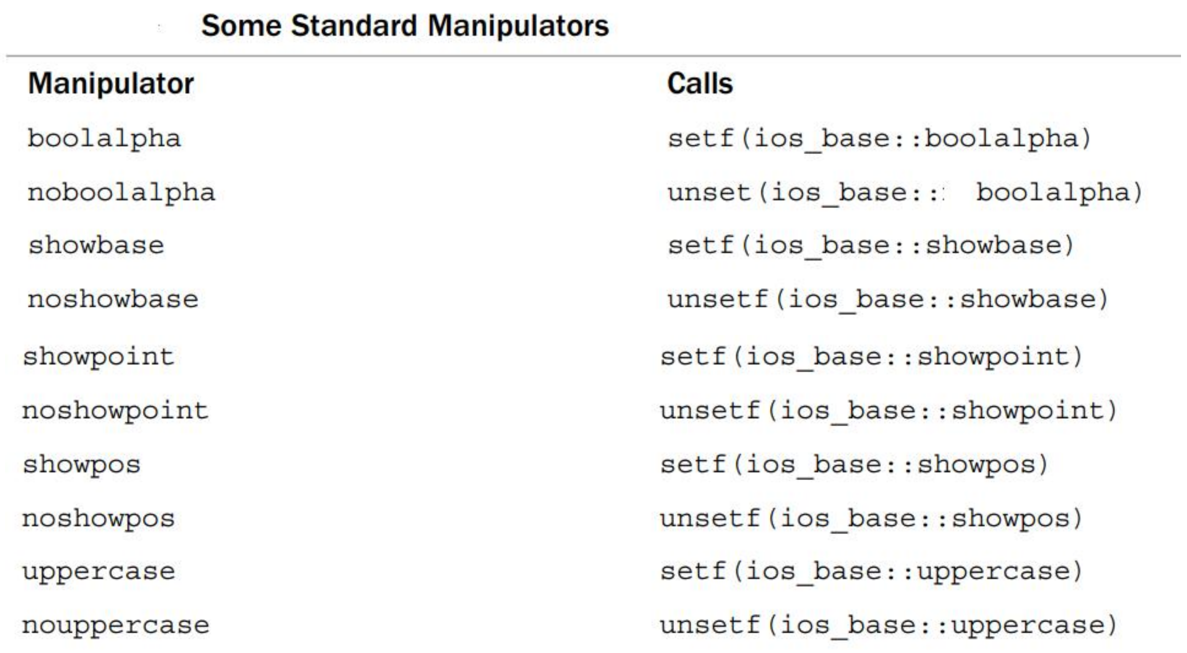


```
56.800000
++456.770000
123.36
3897.67848
```

precision of floating number

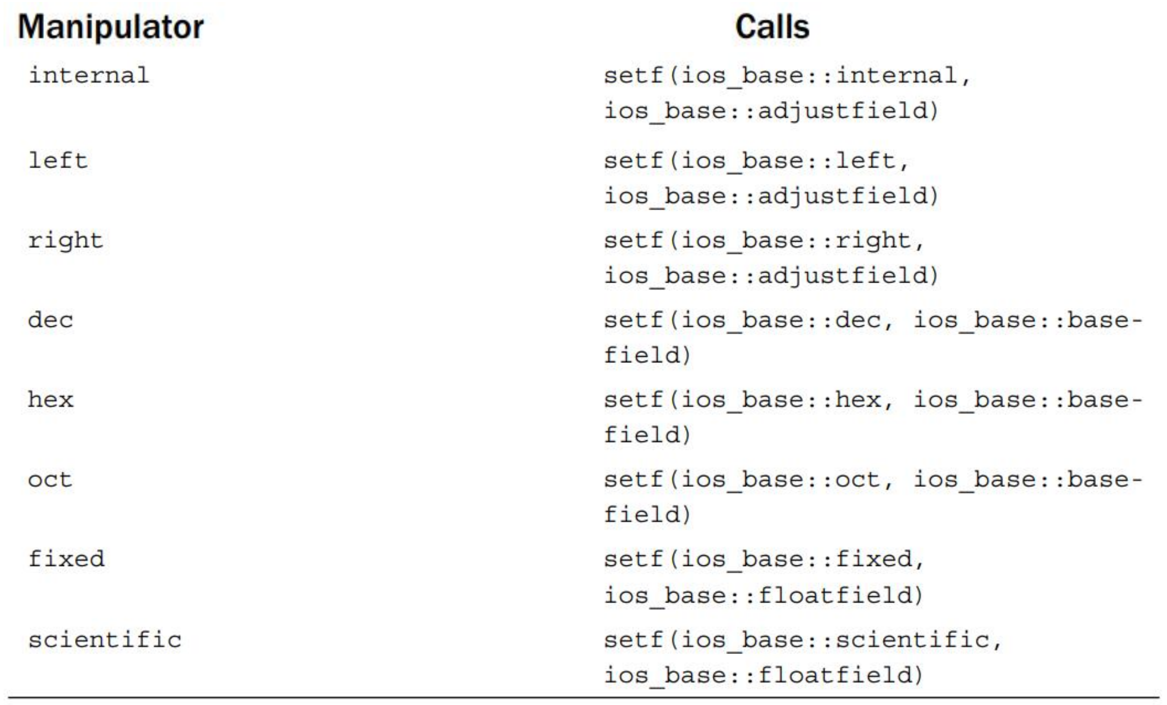
Standard Manipulators

C++ offers several manipulators to invoke **setf()**,automatically supplying the **right arguments**.



Some Standard Manipulators

Manipulator	Calls
`boolalpha`	`setf(ios_base::boolalpha)`
`noboolalpha`	`unset(ios_base:: boolalpha)`
`showbase`	`setf(ios_base::showbase)`
`noshowbase`	`unsetf(ios_base::showbase)`
`showpoint`	`setf(ios_base::showpoint)`
`noshowpoint`	`unsetf(ios_base::showpoint)`
`showpos`	`setf(ios_base::showpos)`
`noshowpos`	`unsetf(ios_base::showpos)`
`uppercase`	`setf(ios_base::uppercase)`
`nouppercase`	`unsetf(ios_base::uppercase)`



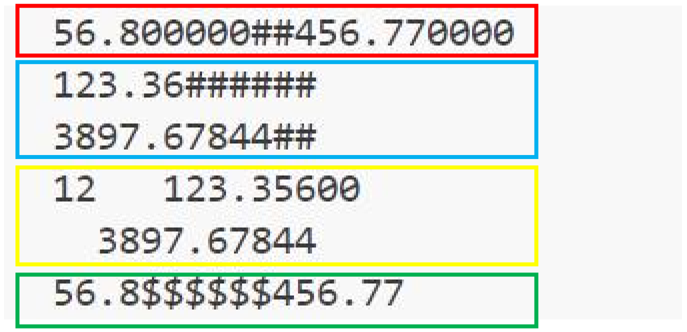
Manipulator	Calls
`internal`	`setf(ios_base::internal, ios_base::adjustfield)`
`left`	`setf(ios_base::left, ios_base::adjustfield)`
`right`	`setf(ios_base::right, ios_base::adjustfield)`
`dec`	`setf(ios_base::dec, ios_base::base-field)`
`hex`	`setf(ios_base::hex, ios_base::base-field)`
`oct`	`setf(ios_base::oct, ios_base::base-field)`
`fixed`	`setf(ios_base::fixed, ios_base::floatfield)`
`scientific`	`setf(ios_base::scientific, ios_base::floatfield)`

- ***Using iomanip manipulators***

  #include <iomanip>

  1. setw(p) 2. setfill(ch) 3. setprecision(d)

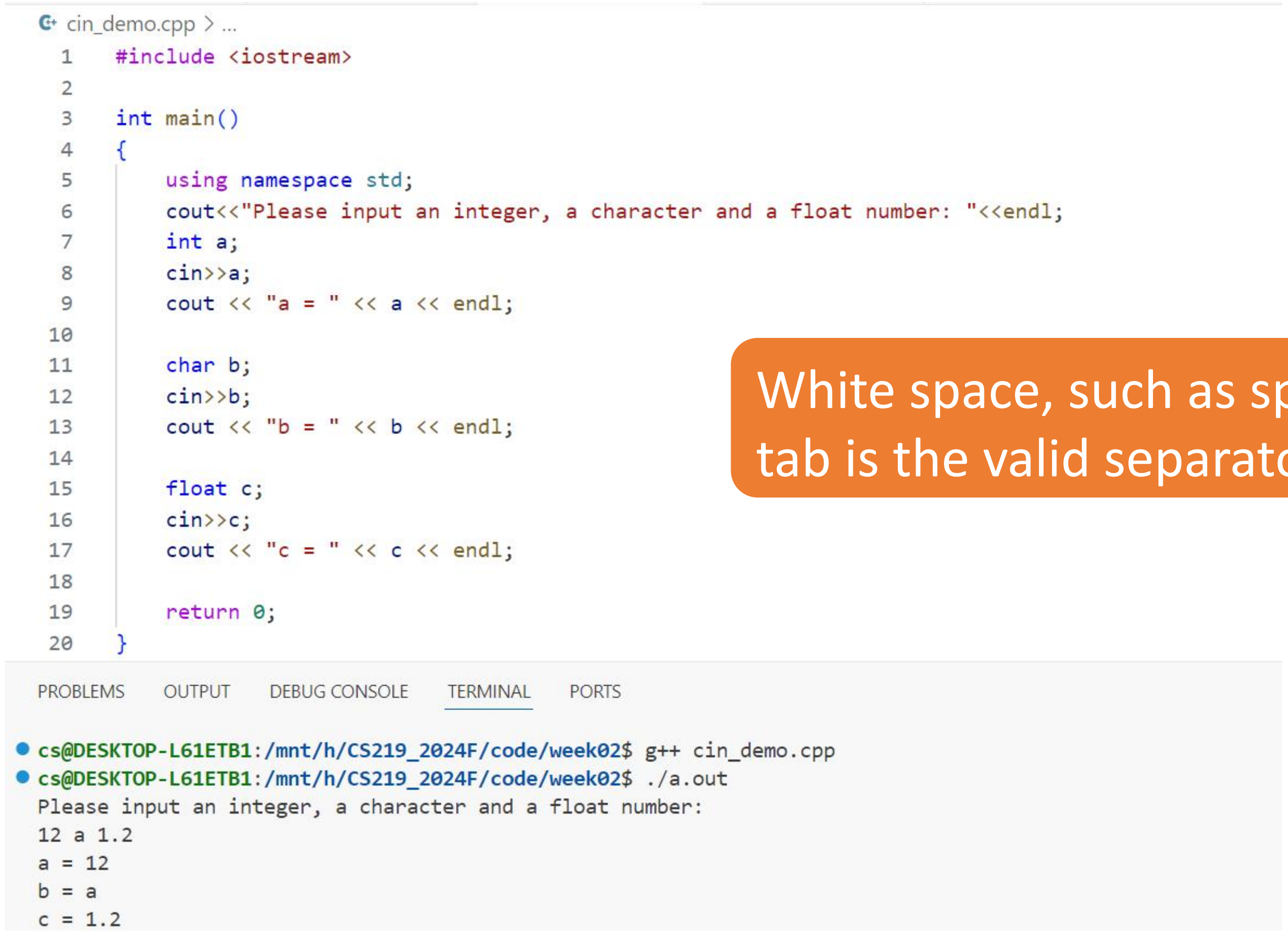
```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
   cout.setf(ios_base::fixed, ios_base::floatfield);
   cout << 56.8 << setw(12) << setfill('#') << 456.77 << endl;

   cout << left;
   cout << setw(12) << setprecision(2) << 123.356 << endl;
   cout << setw(12) << setprecision(5) << 3897.6784385 << endl;

   cout << right;
   cout << 12 << setw(12) << setfill(' ') << 123.356 << endl;
   cout << setw(12) << setfill(' ') << 3897.6784385 << endl;

   cout.unsetf(ios_base::fixed);
   cout << 56.8 << setw(12) << setfill('$') << 456.77 << endl;

   return 0;
}
```



```
56.800000##456.770000
123.36######
3897.67844##
12   123.35600
  3897.67844
56.8$$$$$$$456.77
```

2.3.4. cin

cin >> variable1 [>> variable2 >> ...variable n];



```cpp
#include <iostream>

int main()
{
    using namespace std;
    cout<<"Please input an integer, a character and a float number: "<<endl;
    int a;
    cin>>a;
    cout << "a = " << a << endl;

    char b;
    cin>>b;
    cout << "b = " << b << endl;

    float c;
    cin>>c;
    cout << "c = " << c << endl;

    return 0;
}
```

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ g++ cin_demo.cpp
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week02$ ./a.out
Please input an integer, a character and a float number:
12 a 1.2
a = 12
b = a
c = 1.2
```

White space, such as space, new line and tab is the valid separator.

3 Exercises

Exercise 1. Compile and run the following program, Output the result.

You need to explain the reason to a SA to pass the test.

```c
#include <stdio.h>

int main()
{
    char a = 127;
    unsigned char b = 0xff;
    unsigned char c = 0;
    int d = 65;

    a++;
    b++;
    c--;
    printf("a=%d\nb=%d\nc=%d\nd=%c\n",a,b,c,d);

    return 0;
}
```

Exercise 2. Run the following source code and explain the result.

You need to explain the reason to a SA to pass the test.

```
#include <iostream>
using namespace std;

int main()
{
    cout << fixed;
    float f1 = 1.0f;
    cout<<"f1 = "<<f1<<endl;

    float a = 0.1f;
    float f2 = a+a+a+a+a+a+a+a+a+a;
    cout<<"f2 = "<<f2<<endl;

    if(f1 == f2)
        cout << "f1 = f2" << endl;
    else
        cout << "f1 != f2" << endl;

    return 0;
}
```

Exercise 3. Run the following source code and explain the result.

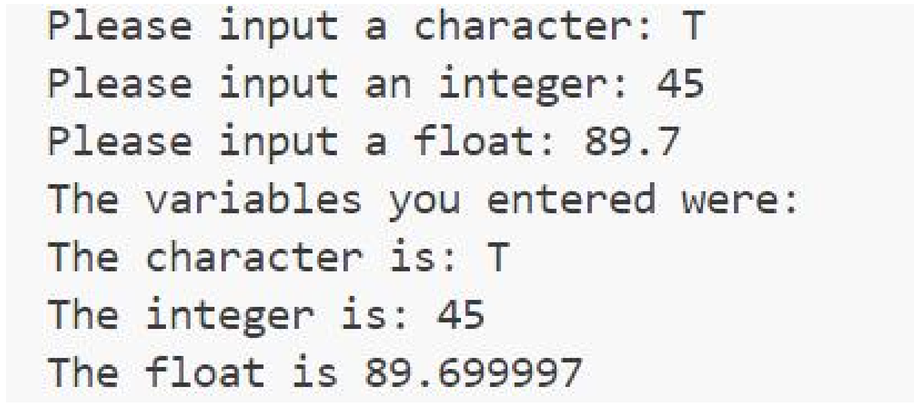
- Why the value of a and b are not equal?
- Explain the division operation with different types.

You need to explain the reason to a SA to pass the test.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int a, b;
    double c, d;

    a = 19.99 + 21.99;
    b = (int)19.99 + (int)21.99;
    c = 23 / 8;
    d = 23 / 8.0;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;
    cout << "d = " << d << endl;
    cout << "0/0= " << 0/0 << endl;

    return 0;
}
```

Exercise 4. Write a **C** program that asks the user to enter an integer value, a character, and a float value and then print them out. A sample run should look like this:

You should use **scanf** and **printf** functions for input and output.



```
Please input a character: T
Please input an integer: 45
Please input a float: 89.7
The variables you entered were:
The character is: T
The integer is: 45
The float is 89.699997
```

What happens when you are prompted to enter an integer, but you enter a float?