Welcome to Week 1

Updated Information

- The original Office Hours
  (By the end of this week (Week 1)):
  1PM - 2PM on Mondays and Wednesdays.
- The new Office Hours
  (Starting from next week (Week 2)):
  9 AM - 10AM on Mondays.
  10AM - 11AM on Wednesdays.
  1PM - 2PM on Fridays.

Waiting List Strategy

- If you are on the waiting list, all the students in the waiting list will enrolled in the class by the end of Week 2.

Learning Tips

Learning Tips

- Understand the Fundamentals: Understand the concepts, do not just memorize the code.
  (in exam, you need to distinuish different errors)

Learning Tips

- Understand the Fundamentals: Understand the concepts, do not just memorize the code.
  (in exam, you need to distinuish different errors)
- Practice Regularly: Consistent practice is key to mastering programming.
  (Reading and Writing code)

Learning Tips

- Understand the Fundamentals: Understand the concepts, do not just memorize the code.
  (in exam, you need to distinuish different errors)
- Practice Regularly: Consistent practice is key to mastering programming.
  (Reading and Writing code)
- Specific for exam: Always try to write code by hand first, as you will need to do so in the exam.

What we learn last week

What we learn last week

- function `main`

What we learn last week

- function `main`
- `#include <iostream>`

What we learn last week

- function `main`
- `#include <iostream>`
- `std::cout` and `std::endl`

What we learn last week

- function `main`
- `#include <iostream>`
- `std::cout` and `std::endl`
- comments `//` and `/* ... */`

What we learn last week

- function `main`
- `#include <iostream>`
- `std::cout` and `std::endl`
- comments `//` and `/* ... */`
- namespace `std::` and `using namespace std;`

What we learn last week

- function `main`
- `#include <iostream>`
- `std::cout` and `std::endl`
- comments `//` and `/* ... */`
- namespace `std::` and `using namespace std;`
- escape characters `\n` , `\t` and `\\` and so on.

Section 5: Variables and Data Types

types

In C++ everything has a type. Basic types include:

- **int**: store integers, without decimals, such as 123 or -123
- **double**: store double-precision floating-point numbers, such as 3.14 or -0.001
- **float**: store single-precision floating-point numbers, such as 3.14 or -0.001
- **char**: store single characters, such as 'a' or 'Z'
- **bool**: store boolean values, either true or false

variables

A variable is a named location in memory that stores a value of a specific type. You must declare a variable before using it, specifying its type and name. For example:

```
int age; // declares an integer variable named age
double pi; // declares a double variable named pi
char grade; // declares a char variable named grade
bool isStudent; // declares a bool variable named isStude
```

Declaring and defining variables

- Declaration and defining variables: Specifies the type and name of the variable. It tells the compiler to allocate memory for the variable.

Declaring and defining variables

- Declaration and defining variables: Specifies the type and name of the variable. It tells the compiler to allocate memory for the variable.

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

int main() {
    int i;

    i = 1;

    cout << i << endl;

    return 0;
}
```

Common Errors (building errors)

Using a variable before it is declared

Common Errors (building errors)

Using a variable before it is declared

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

int main() {
    // int i;

    i = 1;

    cout << i << endl;

    return 0;
}
```

Common Errors (building errors)

- Declaring a variable multiple times with the same name in the same scope

Common Errors (building errors)

- Declaring a variable multiple times with the same name in the same scope

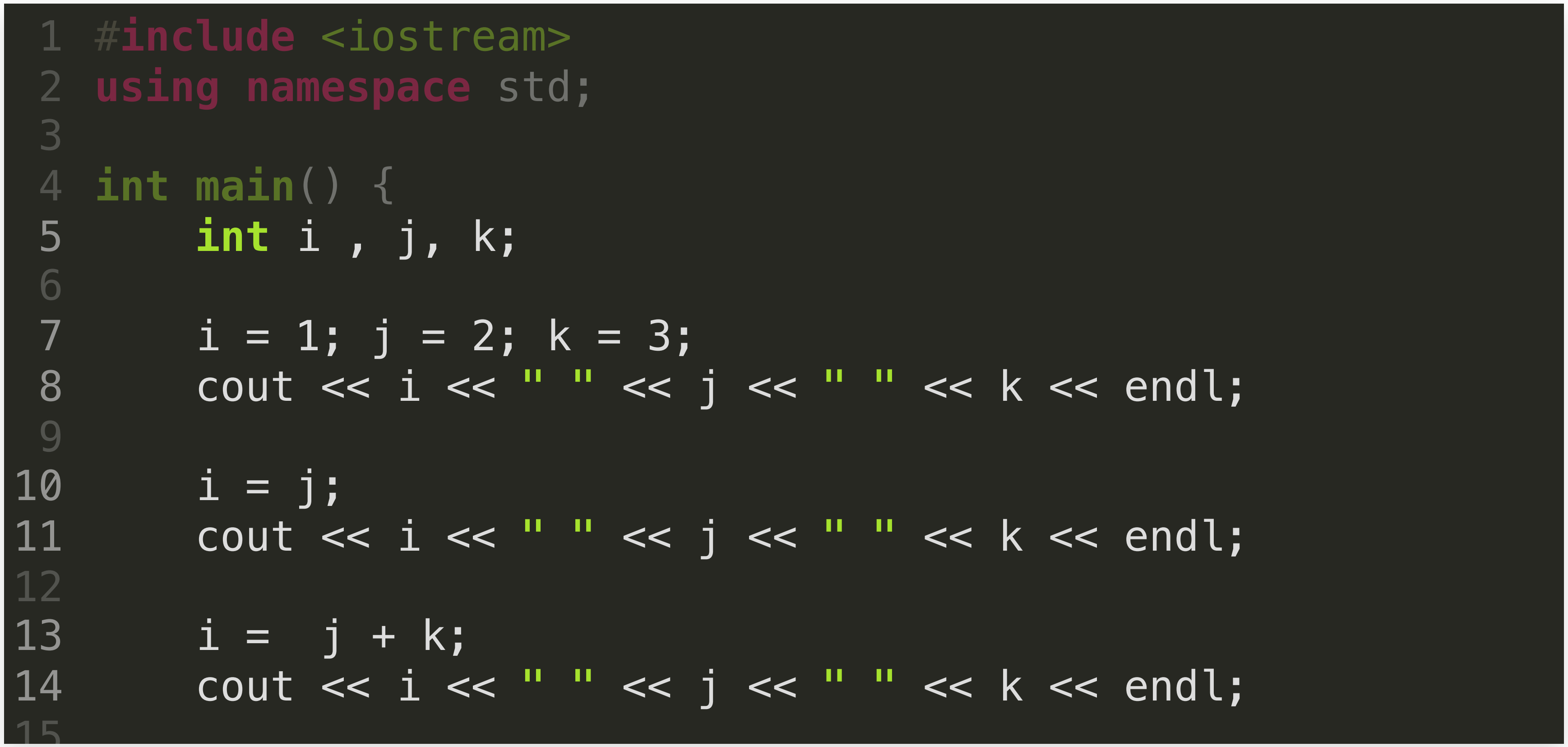
```
#include <iostream>
using namespace std;

int main() {
    int i;

    int i;

    cout << i << endl;

    return 0;
}
```

The assignment operator

- The assignment operator (=) is used to assign a value to a variable.

The assignment operator

- The assignment operator (=) is used to assign a value to a variable.



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

int main() {
    int i , j, k;

    i = 1; j = 2; k = 3;
    cout << i << " " << j << " " << k << endl;

    i = j;
    cout << i << " " << j << " " << k << endl;

    i =  j + k;
    cout << i << " " << j << " " << k << endl;
```

Initializing variables

- Initializing a variable means assigning it an initial value at the time of declaration.
- Common errors **(undefined behavior)**: using a variable before it is initialized.

Initializing variables

- Initializing a variable means assigning it an initial value at the time of declaration.
- Common errors (**undefined behavior**): using a variable before it is initialized.

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

int main() {
    int i; // declaration without initialization

    cout << i << endl; // using uninitialized variable (unc

    i = 5; // initialization

    return 0;
}
```

Naming rules and conventions

- Variable names must start with a letter (a-z, A-Z) or an underscore (_), followed by letters, digits (0-9), or underscores.

Naming rules and conventions

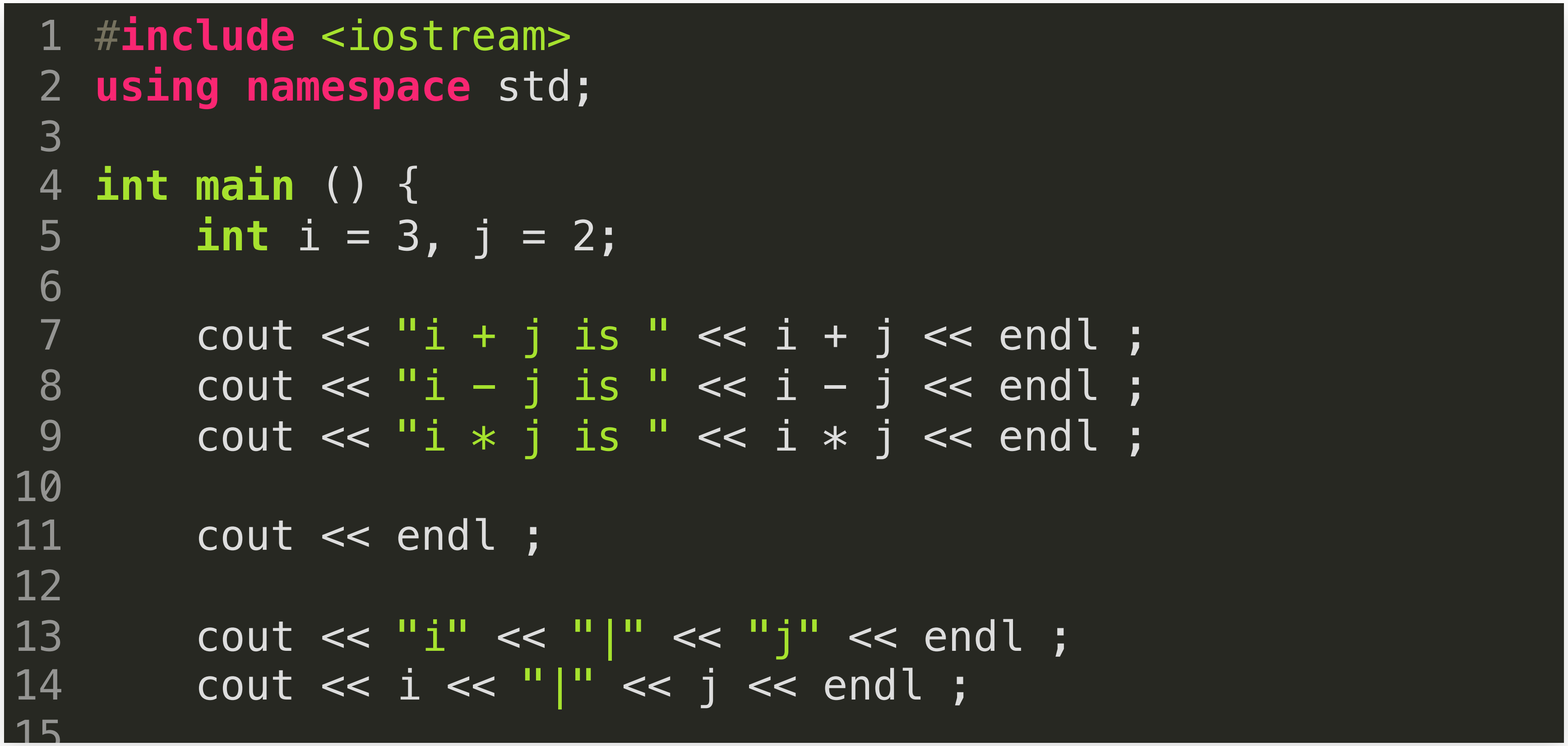
- Variable names must start with a letter (a-z, A-Z) or an underscore (_), followed by letters, digits (0-9), or underscores.

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

int main() {
    int _i = 0;
    int i_ = 1;
    int i1 = 2;
    int 1i = 3;
    cout << _i << endl;
    return 0;
}
```

Naming rules and conventions

- Variable names must start with a letter (a-z, A-Z) or a underscore (_), followed by letters, digits (0-9), or underscores.
- Variable names are case-sensitive (e.g., myVar and myvar are different variables).
- common convections:
    - ALL_CAPS for constants
    - words_which_May_or_may_not_start_with_a_Capital_separated_by_unders for variables and functions.
    - UpperCamelCase
    - lowerCamelCase

+, -, * , +=, -= , *=



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

int main () {
    int i = 3, j = 2;

    cout << "i + j is " << i + j << endl ;
    cout << "i - j is " << i - j << endl ;
    cout << "i * j is " << i * j << endl ;

    cout << endl ;

    cout << "i" << "|" << "j" << endl ;
    cout << i << "|" << j << endl ;
```

++ and --

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

int main () {
    int i, ppi, ipp;

    i=0 ; ++i;
    cout << "i is " << i << endl ;

    i=0 ; ppi = ++i;
    cout << "i is " << i << ", ppi is " << ppi << endl ;

    i=0 ; ipp = i++;
    cout << "i is " << i << ", ipp is " << ipp << endl ;
```

Common Error (undefined behavior)

Using ++ or -- more than once in the same expression.

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

int main () {
    int i, ppi, ipp;

    i=0 ; ppi = ++i + ++i;
    cout << "i is " << i << ", ppi is " << ppi << endl ;

    i=0 ; ipp = i++ + i++;
    cout << "i is " << i << ", ipp is " << ipp << endl ;


    return 0;
}
```

Integer Division

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

int main () {
    int i, j;

    i = 28;
    j = 10;

    cout << "i / j is " << i / j << endl ;
    cout << "i % j is " << i % j << endl ;

    i = -28;
    j = 10;
    cout << "i / j is " << i / j << endl ;
```

What is int?

- int has a fixed size (usually 4 bytes)
- int can represent integers in a fixed range (usually -2,147,483,648 to 2,147,483,647)
- If you try to store a number outside this range in an int variable, you will get integer overflow or underflow, which leads to **undefined behavior.**

const

- Use const to declare variables whose values should not change.
- Example: const int DAYS_IN_A_WEEK = 7;

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

int main () {
    const int HOURS_IN_A_DAY = 24;
    const int MINUTES_IN_AN_HOUR = 60;

    const int MINUTES_IN_A_DAY = HOURS_IN_A_DAY * MINUTES_I
    cout << "There are " << MINUTES_IN_A_DAY << " minutes i

    return 0;
}
```

Common Error (building error)

Attempting to modify a const variable.

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

int main () {
    const int HOURS_IN_A_DAY = 24;
    HOURS_IN_A_DAY = 25; // Error: cannot modify a const var

    return 0;
}
```

Common Error (building error)

the const type variable must be initialized when it is declared.

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

int main () {
    const int HOURS_IN_A_DAY; // Error: const variable must
    HOURS_IN_A_DAY = 24;

    return 0;
}
```

std::cin

- std::cin is used to take input from the user.

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

int main () {
    int age;
    cout << "Enter your age: ";
    cin >> age;
    cout << "Your age is " << age << endl;

    return 0;
}
```

multiple inputs

- You can take multiple inputs using std::cin.

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

int main () {
    int i1 , i2;
    cout << "Enter two integers separated by a space: ";
    cin >> i1 >> i2;
    cout << "You entered: " << i1 << " and " << i2 << endl;
    cout << "Their sum is " << i1 + i2 << endl;
    return 0;
}
```

bool

- bool is a data type that can hold one of two values: true or false.

```cpp
#include <iostream>
using namespace std;
int main() {
    bool isStudent;
    isStudent = true;
    isStudent = false;

    cout << "Is student: " << isStudent << endl;
    return 0;
}
```

==

- The == operator is used to compare two values.

```cpp
#include <iostream>
using namespace std;
int main() {
    int a = 5;
    int b = 10;
    bool result;
    result = (a == b);
    cout << boolalpha; // print bool as true/false
    cout << a << " == " << b << ": " << result << endl;
    return 0;
}
```

Exercises

write a code that ask the user to input two integers, then output whether they are equal.

```cpp
#include <iostream>
using namespace std;
int main() {
    int a, b;
    cout << boolalpha; // print bool as true/false
    cout << "Enter two integers: ";
    // your code here: to read a and b from user input
    // your code here: to compare a and b, and store the re
    // your code here: to print whether a and b are equal o
    return 0;
}
```

Exercises

write a code that ask the user to input two integers, then output whether they are equal.

```cpp
#include <iostream>
using namespace std;
int main() {
    int a, b;
    cout << boolalpha; // print bool as true/false
    cout << "Enter two integers: ";
    cin >> a >> b;
    bool result = (a == b);
    cout << a << " == " << b << ": " << result << endl;
    return 0;
}
```

Exercises

write a code that ask the user to input an integer, then output whether it is even.

```cpp
#include <iostream>
using namespace std;
int main() {
    int n;
    cout << boolalpha; // print bool as true/false
    cout << "Enter an integer: ";
    // your code here: to read n from user input
    // your code here: to determine whether n is even, and
    // your code here: to print whether n is even or not
    return 0;
}
```

Exercises

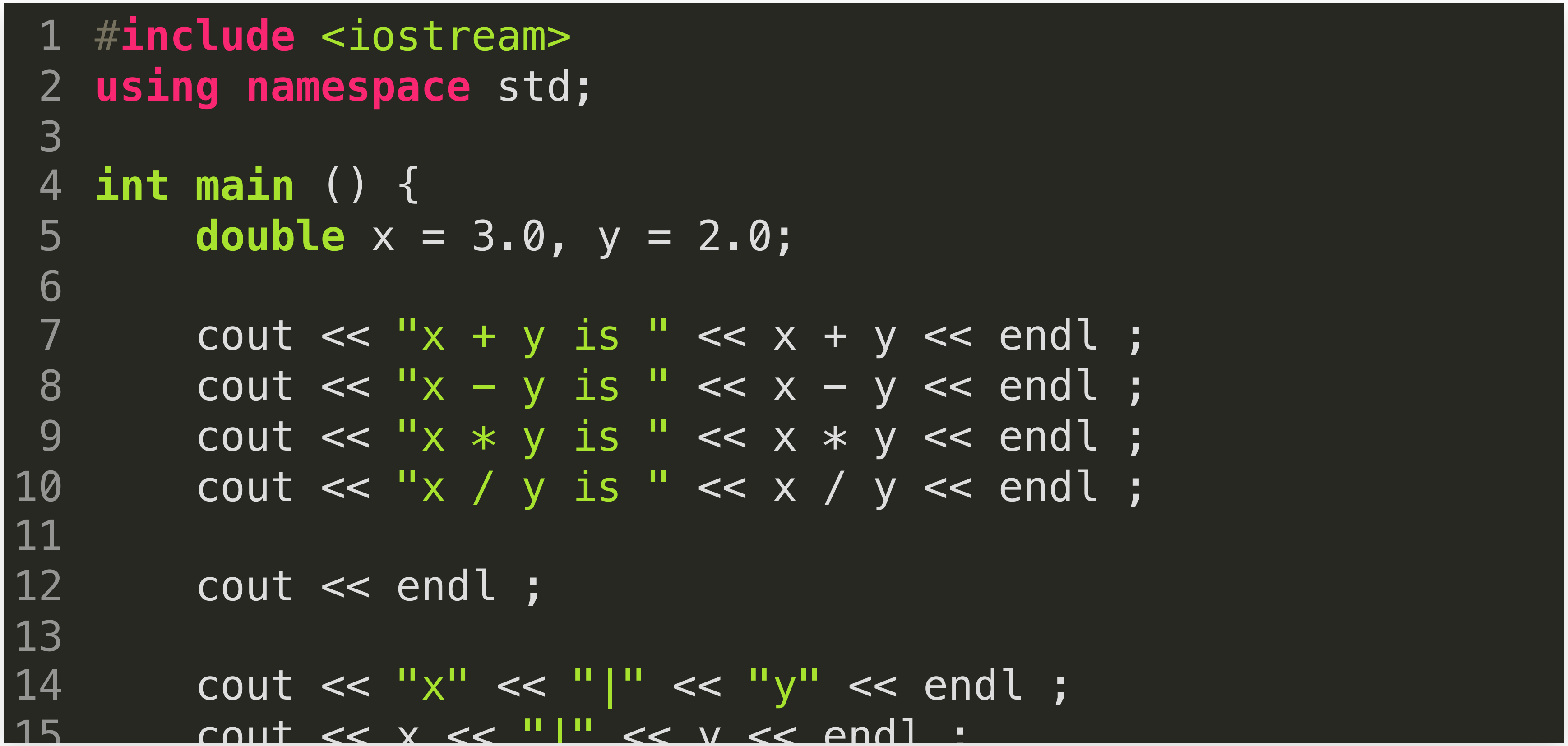
write a code that ask the user to input an integer, then output whether it is even.

```cpp
#include <iostream>
using namespace std;
int main() {
    int n;
    cout << boolalpha; // print bool as true/false
    cout << "Enter an integer: ";
    cin >> n;
    bool isEven = (n % 2 == 0);
    cout << n << " is even: " << isEven << endl;
    return 0;
}
```

double

- double is a data type that can hold floating-point numbers (numbers with decimals).
- "double" means double precision floating point.

+, - , * , / , +=, -=, *=, /=



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

int main () {
    double x = 3.0, y = 2.0;

    cout << "x + y is " << x + y << endl ;
    cout << "x - y is " << x - y << endl ;
    cout << "x * y is " << x * y << endl ;
    cout << "x / y is " << x / y << endl ;

    cout << endl ;

    cout << "x" << "|" << "y" << endl ;
    cout << x << "|" << y << endl ;
```

Common Errors

double-division by zero gives undefined behavior.

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

int main () {
    double x = 3.0, y = 0.0;

    cout << "x / y is " << x / y << endl ;

    return 0;
}
```

Type Casting

- Type casting is the process of converting a value from one data type to another.
- You can use static_cast to perform type casting in C++.

Type Casting

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

int main () {
    int i = 3;
    double x;

    x = static_cast<double>(i);
    cout << "x is " << x << endl ;

    i = static_cast<int>(x + 0.99);
    cout << "i is " << i << endl ;

    return 0;
}
```

Mixed division

- When you divide an int by a double, the int is automatically converted to a double before the division.
- When you divide a double by an int, the int will also be automatically converted to a double before the division.

Mixed division

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

int main () {
    int a = 5;
    double b = 2.0;

    cout << "a / b is " << a / b << endl ;
    cout << "b / a is " << b / a << endl ;

    return 0;
}
```

Mixed division

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

int main () {
    int i1 = 5 , i2 = 2;
    cout << i1 / i2 << endl ;
    cout << static_cast<double>(i1) / i2 << endl ;
    cout << i1 / static_cast<double>(i2) << endl ;
    cout << static_cast<double>(i1)
    / static_cast<double>(i2) << endl ;

    return 0;
}
```

cmath

- cmath is a header file that contains mathematical functions.
- Some common functions in cmath are:
    - sqrt(x): returns the square root of x
    - pow(x, y): returns x raised to the power of y
    - abs(x): returns the absolute value of x
    - ceil(x): returns the smallest integer greater than or equal to x
    - floor(x): returns the largest integer less than or equal to x
    - round(x): returns the nearest integer to x

cmath

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

int main () {
    double x = 3.5;

    cout << "sqrt(x) is " << sqrt(x) << endl ;
    cout << "pow(x, 2) is " << pow(x, 2) << endl ;
    cout << "abs(-x) is " << abs(-x) << endl ;
    cout << "ceil(x) is " << ceil(x) << endl ;
    cout << "floor(x) is " << floor(x) << endl ;
    cout << "round(x) is " << round(x) << endl ;
    return 0;
}
```

what is a double ?

precision of double

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

int main () {
    double x = 1.0 / 3.0;

    cout << x << endl ;
    cout << setprecision(2) << x << endl ;
    cout << setprecision(5) << x << endl ;
    cout << setprecision(10) << x << endl ;
    cout << setprecision(20) << x << endl ;

    return 0;
}
```

Example

```cpp
#include <iostream>
using namespace std;
int main() {
    double i = 4.0, j = 3.0;
    cout << boolalpha; // print bool as true/false
    double k = i / j - 1.0 - 1.0 / 3.0;
    cout << "k is " << k << endl;
    cout << "k == 0.0: " << (k == 0.0) << endl;
    return 0;
}
```

unsigned int

- unsigned int is a data type that can hold only non-negative integers (0 and positive integers).
- The range of unsigned int is usually from 0 to 4,294,967,295 (2^32 - 1).
- If you try to store a negative number in an unsigned int variable, you will get integer underflow, which leads to **undefined behavior.**

unsigned int

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

int main () {
    unsigned int u = 0;

    cout << "u is " << u << endl ;

    u = u - 1; // underflow

    cout << "u is " << u << endl ;

    return 0;
}
```

static_cast

- `static_cast<unsigned int>(x)` converts `x` to an `unsigned int`.
  - If `x` is non-negative, the value stays the same.
  - If `x` is negative, C++ adds 2^{32} (on a 32-bit system) and the result becomes a large positive number.

static_cast

- `static_cast<int>(x)` converts `x` to an `int`.
  - If `x` is within the range of `int`, the value stays the same.
  - If `x` is larger than the maximum `int`, C++ subtracts 2^{32} (on a 32-bit system), and the result is a negative number.

Example

```cpp
#include <iostream>
using namespace std;
int main() {
    int i = -1;
    unsigned int u = static_cast<unsigned int>(i);
    cout << "i is " << i << endl;
    cout << "u is " << u << endl;

    unsigned int u2 = 4294967295; // 2^32 - 1
    int i2 = static_cast<int>(u2);
    cout << "u2 is " << u2 << endl;
    cout << "i2 is " << i2 << endl;

    return 0;
}
```

std::size_t

- `std::size_t` is an unsigned integer type.
- It is used to represent the size of objects in bytes.
- It is commonly used for array indexing and loop counters.

char

- char is a data type that can hold a single character.
- char is usually 1 byte (8 bits) in size.
- char can represent characters using ASCII or Unicode encoding.

char

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

int main () {
    char c = 'A';
    int  i = 65;

    cout << "c is " << c << endl ;
    cout << "i is " << i << endl ;

    cout << "static_cast<int>(c) is " << static_cast<int>(
    cout << "static_cast<char>(i) is " << static_cast<char

    return 0;
}
```

char

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

int main () {
    char c = 'A';
    int  i = 65;

    cout << "c is " << c << endl ;
    cout << "i is " << i << endl ;

    cout << "static_cast<int>(c) is " << static_cast<int>(c
    cout << "static_cast<char>(i) is " << static_cast<char>

    return 0;
}
```

- The `static_cast<int>(c)` converts the character 'A' to its ASCII integer value 65.
- The `static_cast<char>(i)` converts the integer 65 back to the character 'A'.

char

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

int main () {
    char c = 'A';

    cout << "c is " << c << endl ;
    cout << "c + 1 is " << c + 1 << endl ;
    cout << "c + 2 is " << c + 2 << endl ;

    return 0;
}
```

char

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

int main () {
    char c = 'A';

    cout << "c is " << c << endl ;
    cout << "c + 1 is " << c + 1 << endl ;
    cout << "c + 2 is " << c + 2 << endl ;

    return 0;
}
```

- When you add an integer to a char, the char is automatically converted to its ASCII integer value before the addition.

static_cast<T> and implicit casting

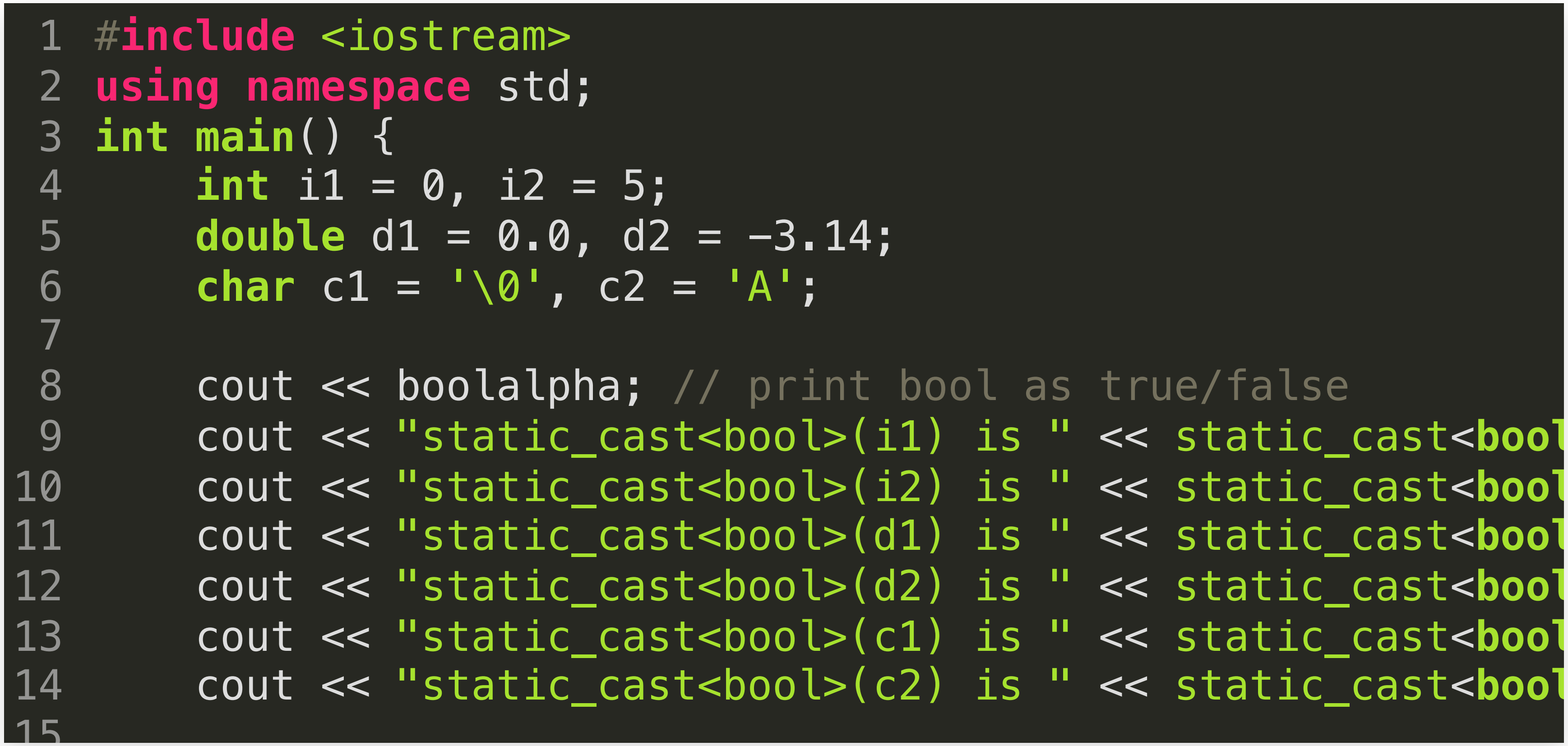
static_cast<T> and implicit casting

- `static_cast<T>(expr)` is used to explicitly convert `expr` to type T.
- Implicit casting is done automatically by the compiler when it is safe to do so.

static_cast<bool>

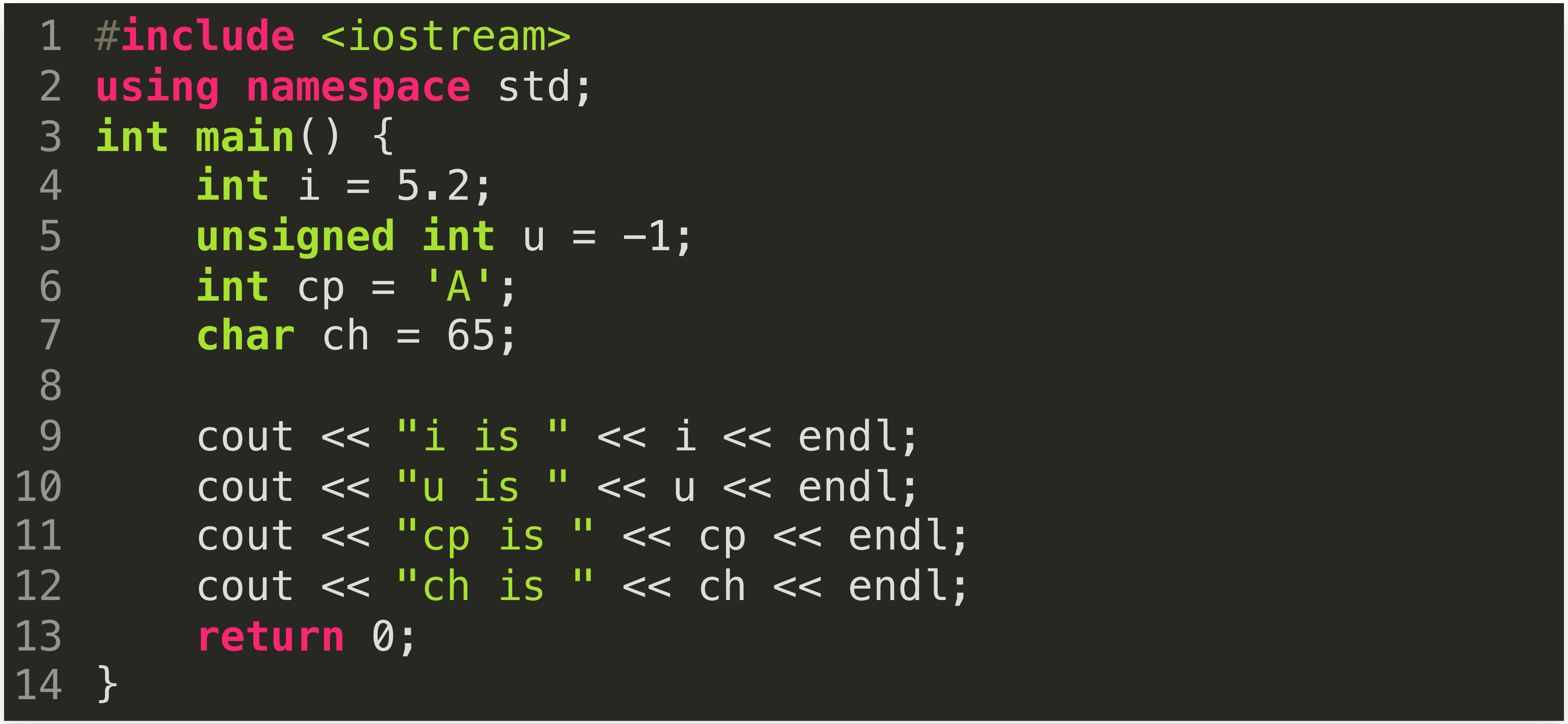
- `static_cast<bool>(x)` converts `x` to a `bool`.
  - If `x` is zero (0 for int, 0.0 for double, '\0' for char), the result is `false`.
  - If `x` is non-zero (any other value), the result is `true`.

static_cast<bool>



```
#include <iostream>
using namespace std;
int main() {
    int i1 = 0, i2 = 5;
    double d1 = 0.0, d2 = -3.14;
    char c1 = '\0', c2 = 'A';

    cout << boolalpha; // print bool as true/false
    cout << "static_cast<bool>(i1) is " << static_cast<bool
    cout << "static_cast<bool>(i2) is " << static_cast<bool
    cout << "static_cast<bool>(d1) is " << static_cast<bool
    cout << "static_cast<bool>(d2) is " << static_cast<bool
    cout << "static_cast<bool>(c1) is " << static_cast<bool
    cout << "static_cast<bool>(c2) is " << static_cast<bool
```

implicit casting



```cpp
#include <iostream>
using namespace std;
int main() {
    int i = 5.2;
    unsigned int u = -1;
    int cp = 'A';
    char ch = 65;

    cout << "i is " << i << endl;
    cout << "u is " << u << endl;
    cout << "cp is " << cp << endl;
    cout << "ch is " << ch << endl;
    return 0;
}
```