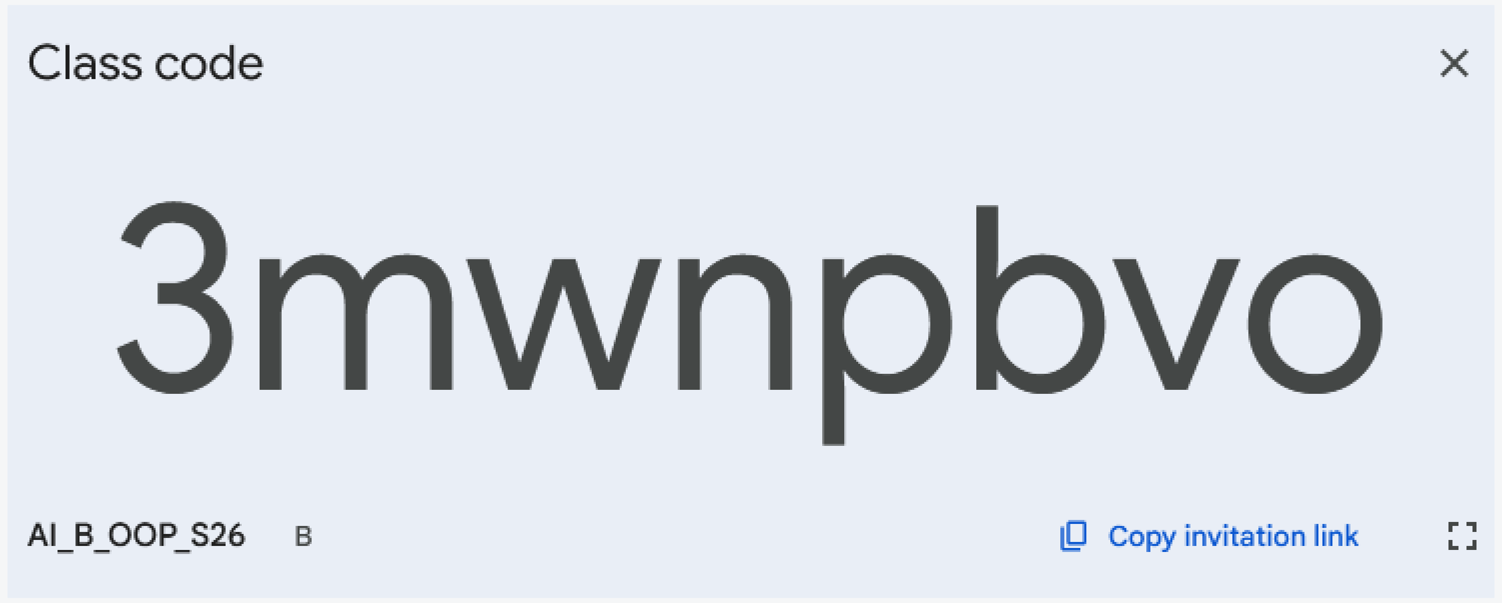
Lecture-1 Introduction

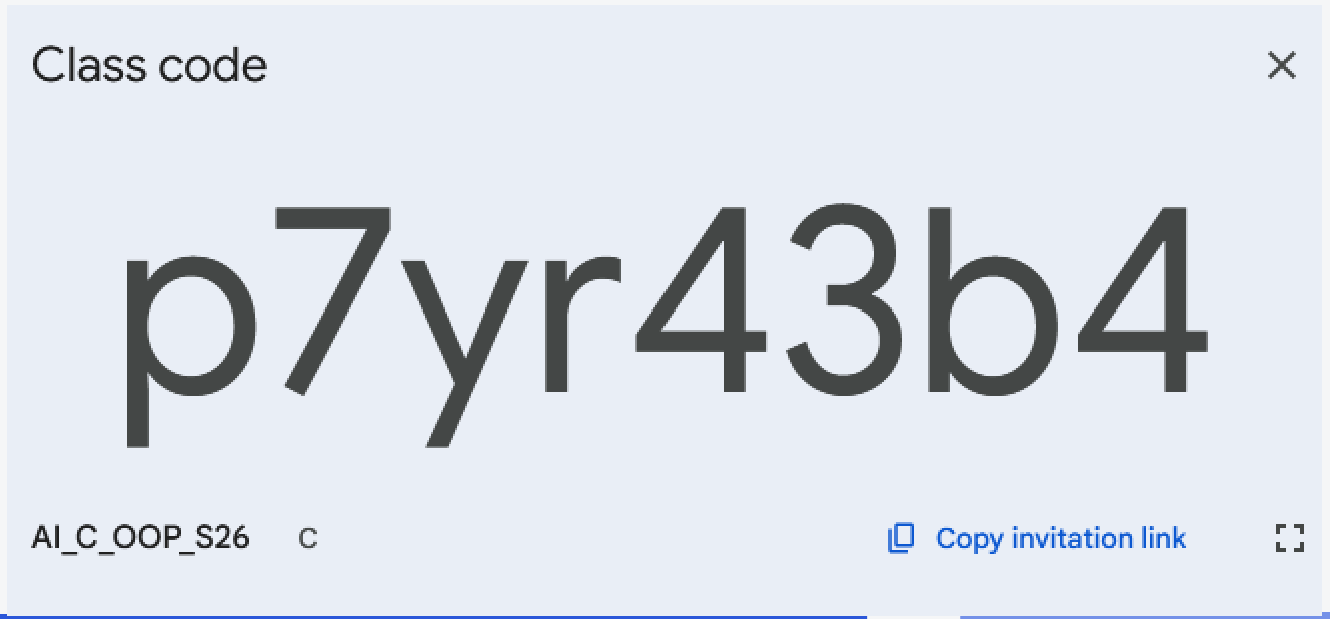
Instructions

- Please join ***Google Classroom*** to get updates.
- Attendance will be marked on ***Google Forms***.
- In case of any query, contact me on my email. (***ishabansatti@gmail.com***)
- Please be punctual in class.


Class code

3mwnpbvo

AI_B_OOP_S26 B


Class code

p7yr43b4

AI_C_OOP_S26 C

Marks Breakup

• Quizzes	(25 = 5 X 5)
• Assignments	(5 = 1 X 5)
• Mid Exam	(30)
• Final Exam	(40)

Instructions

- MCQs based quizzes on google forms.
- You should have a GitHub account to upload your assignment codes.
- All assignments will be followed by a relevant viva.

Tools Required for this course

- CLion(code editor) **https://www.jetbrains.com/clion/**
- GCC C++ compiler and GDB debugger from mingw-w64

  https://code.visualstudio.com/docs/cpp/config-mingw
- Safe exam browser **https://safeexambrowser.org/download_en.html**
- CMake **https://cmake.org/download/**

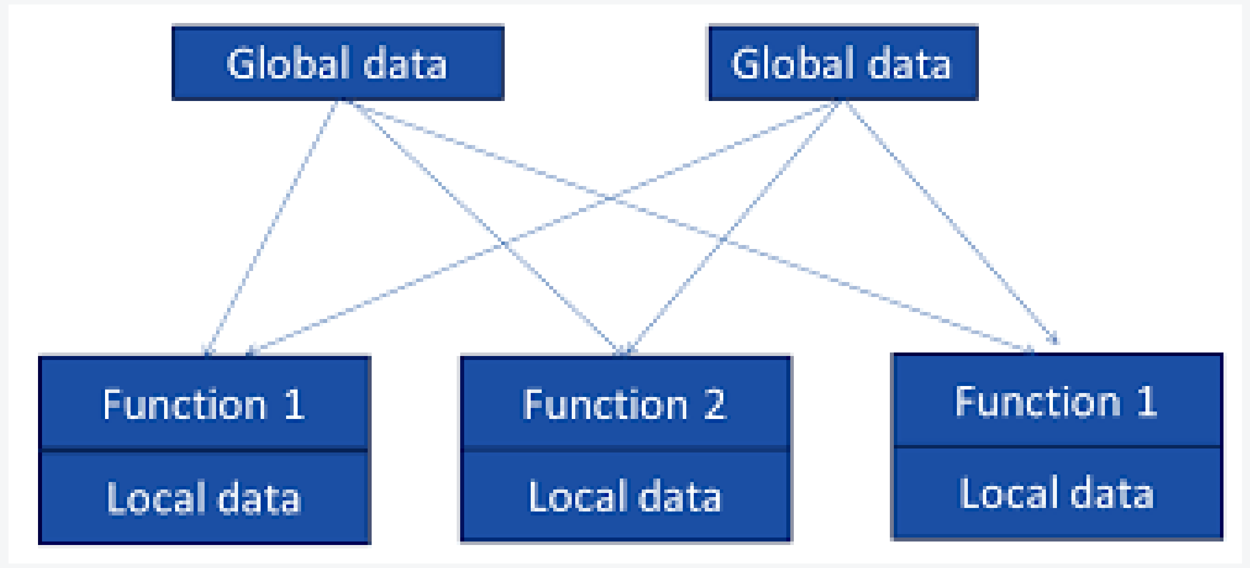
Agenda

01.

Why do we need OOP?

Procedural Programming

- Structures code as a sequence of instructions.
- Organized into reusable blocks called **functions**, which are called in a specific order to perform tasks.
- Key languages include C.



Issues with Procedural Paradigm

1. Unrestricted access

2. Poor real-world mapping

3. Inextensibility

Object Oriented Programming

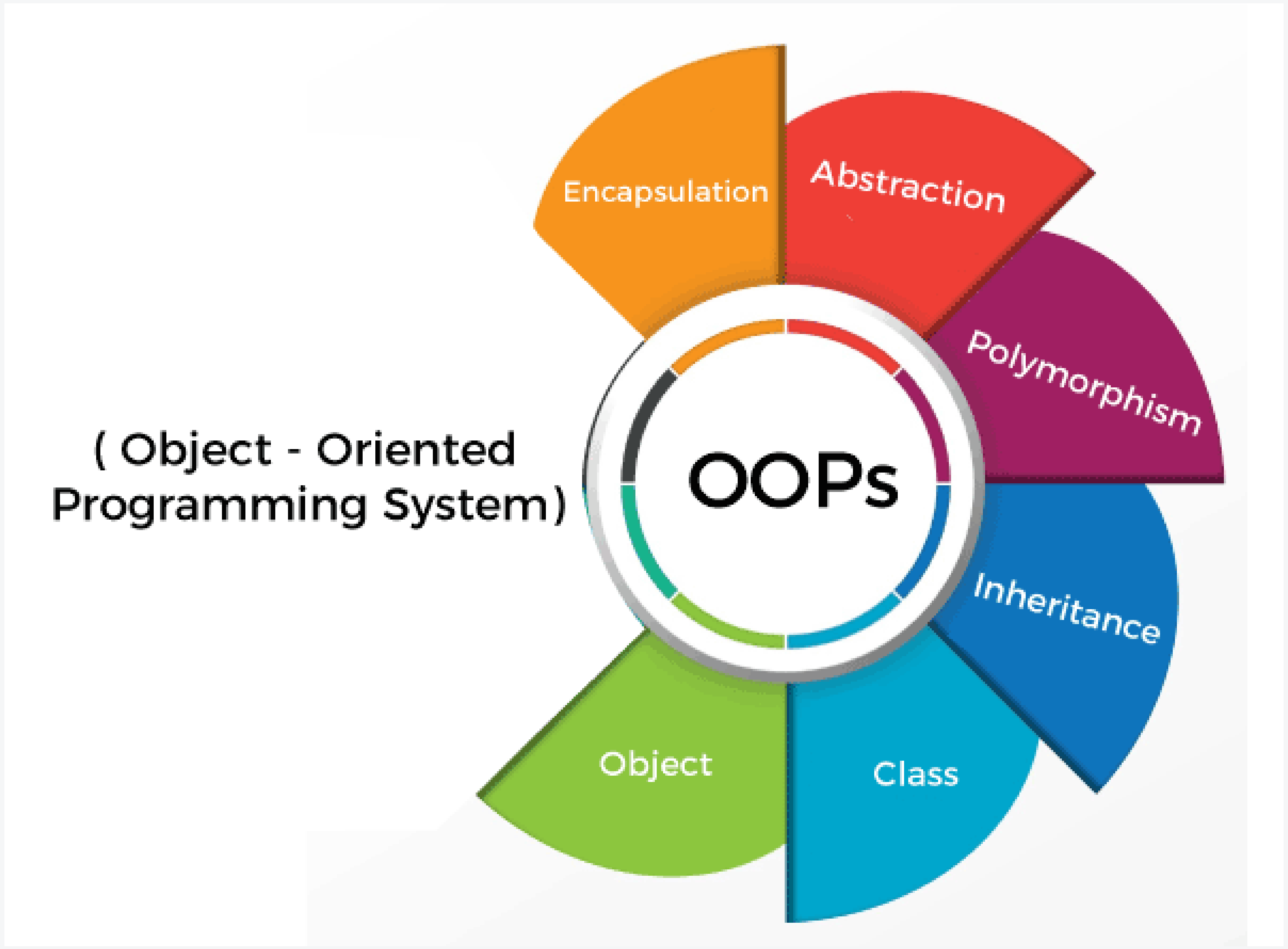
- Combines data (data items) and relevant functions (member functions) in a single unit (object).

Issues with Procedural Paradigm

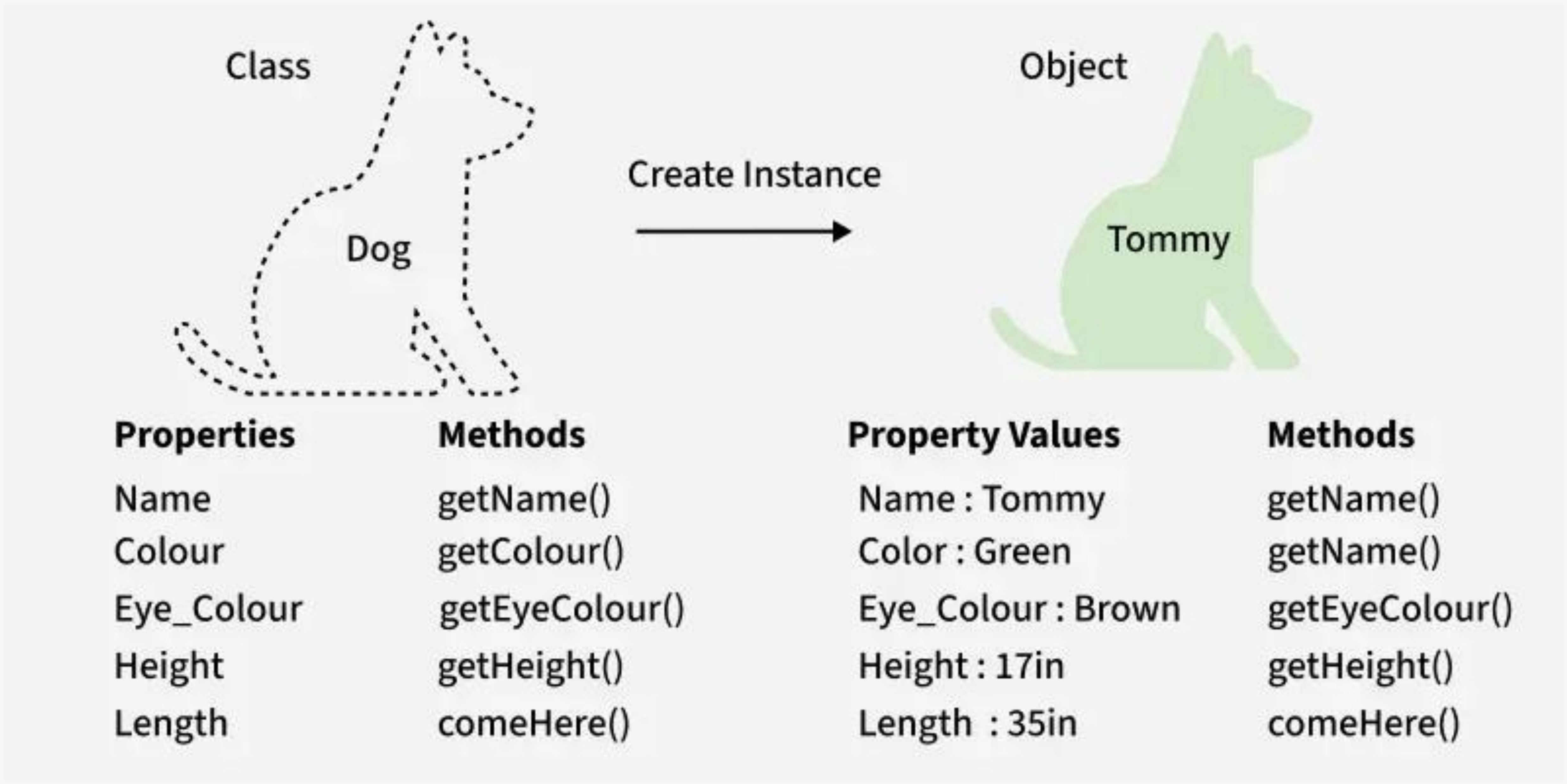
1. Unrestricted access
2. Poor real-world mapping
3. Inextensibility

02.

Characteristics of OOP


(Object - Oriented
Programming System)
OOPs
Encapsulation
Abstraction
Polymorphism
Inheritance
Class
Object

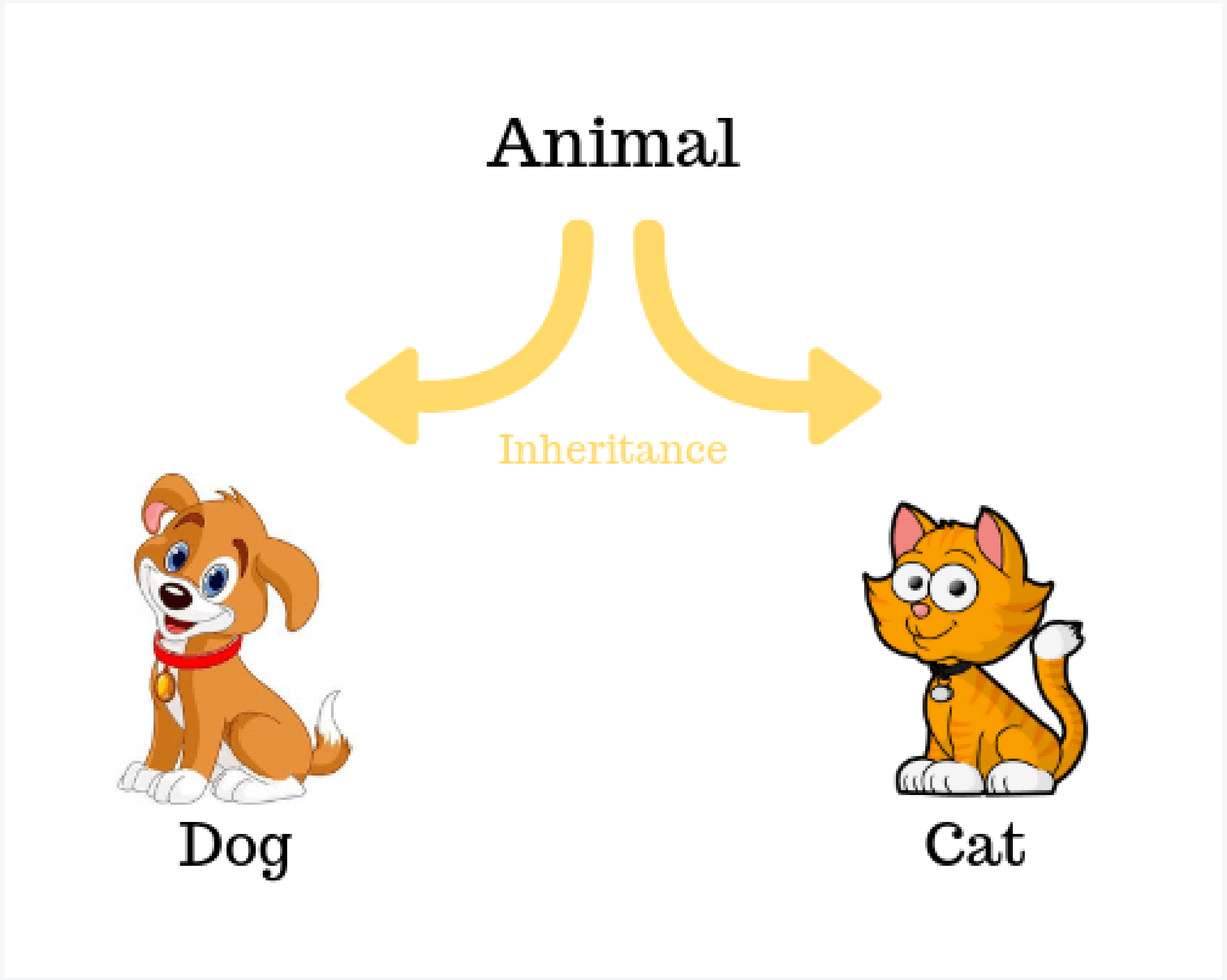
Class and Object



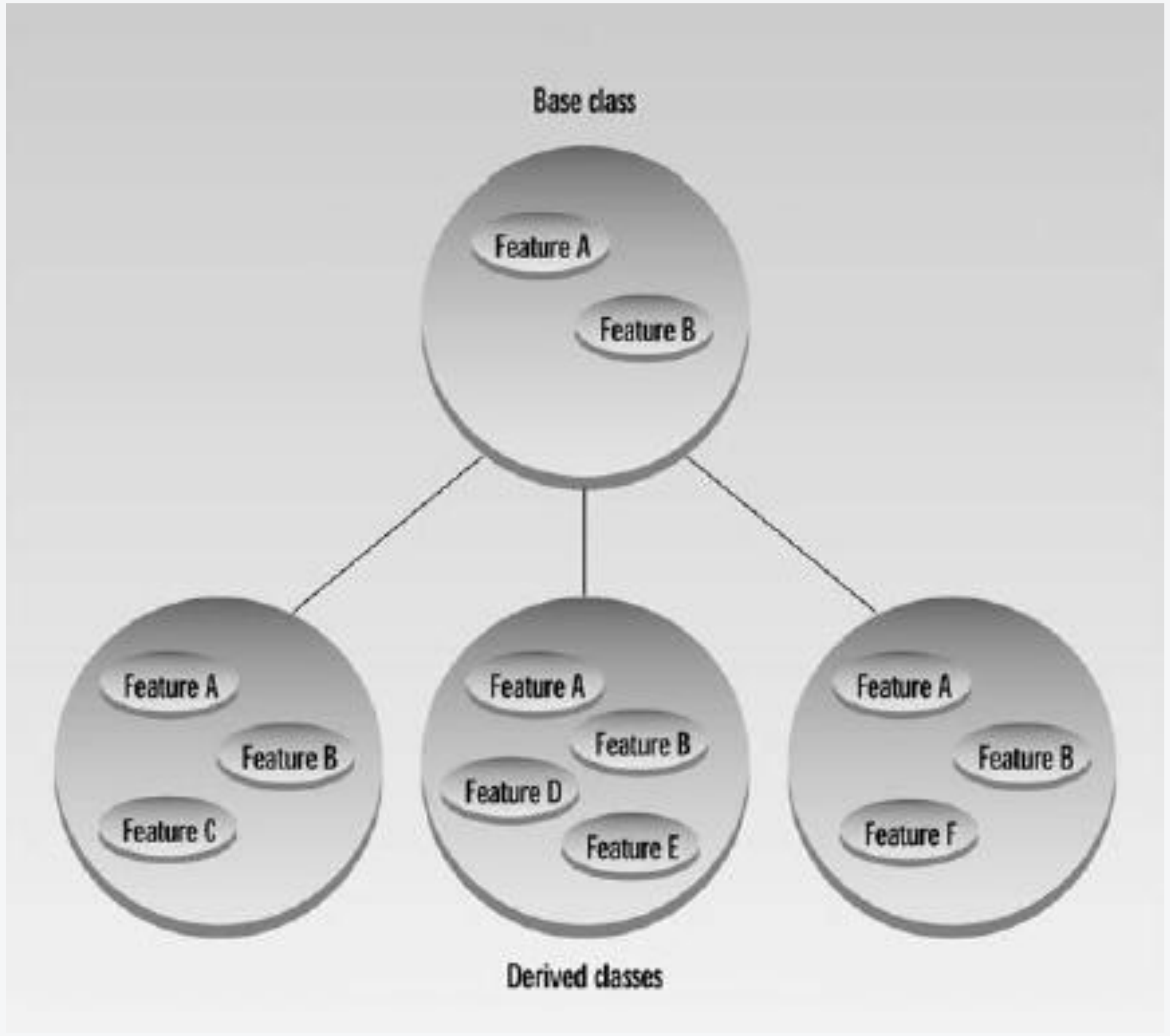
Properties	Methods
Name	getName()
Colour	getColour()
Eye_Colour	getEyeColour()
Height	getHeight()
Length	comeHere()

Property Values	Methods
Name : Tommy	getName()
Color : Green	getName()
Eye_Colour : Brown	getEyeColour()
Height : 17in	getHeight()
Length : 35in	comeHere()

Inheritance and Reusability



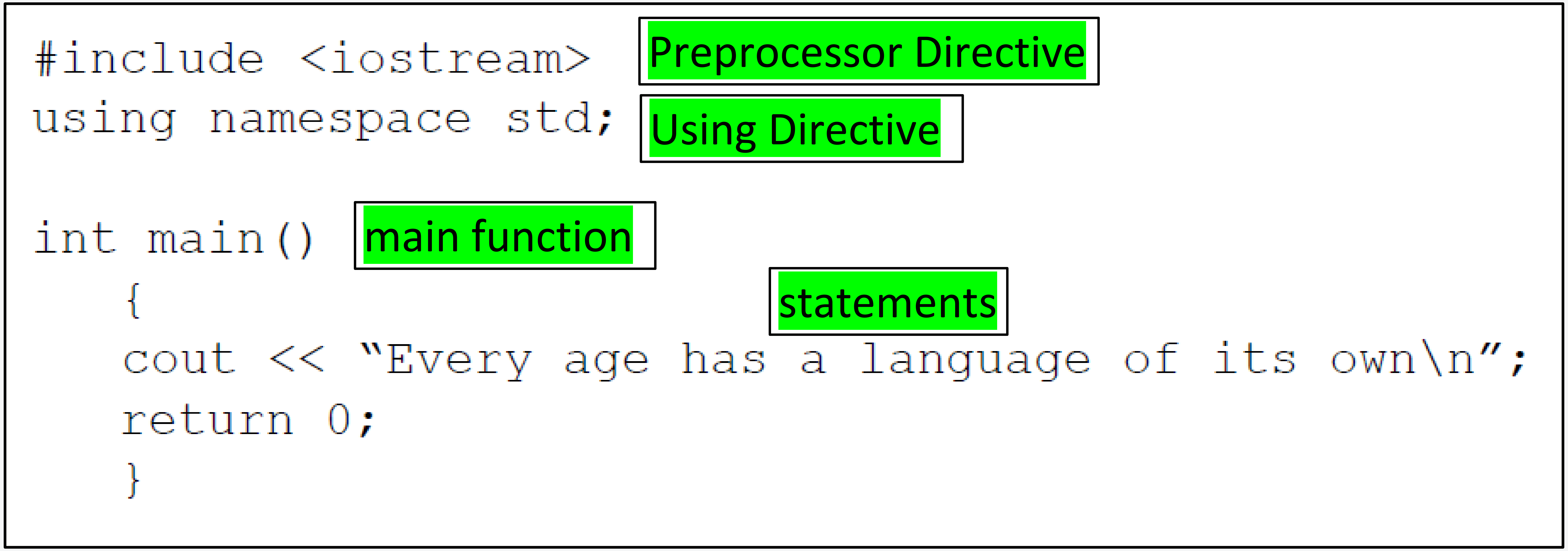
Inheritance and Reusability



03.

A Basic C++ Program

A Basic C++ Program

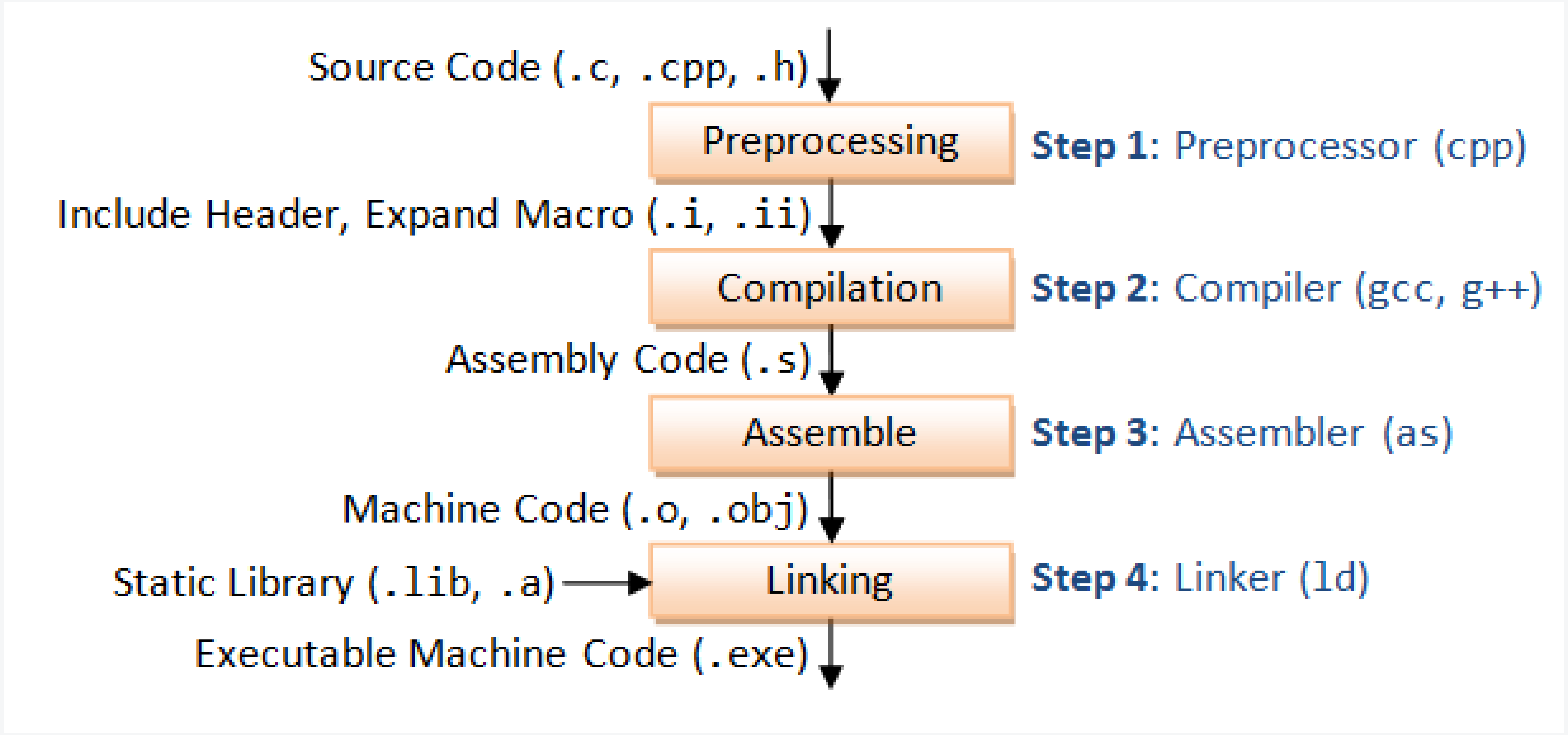


```
#include <iostream>
using namespace std;

int main()
    {
    cout << "Every age has a language of its own\n";
    return 0;
    }
```

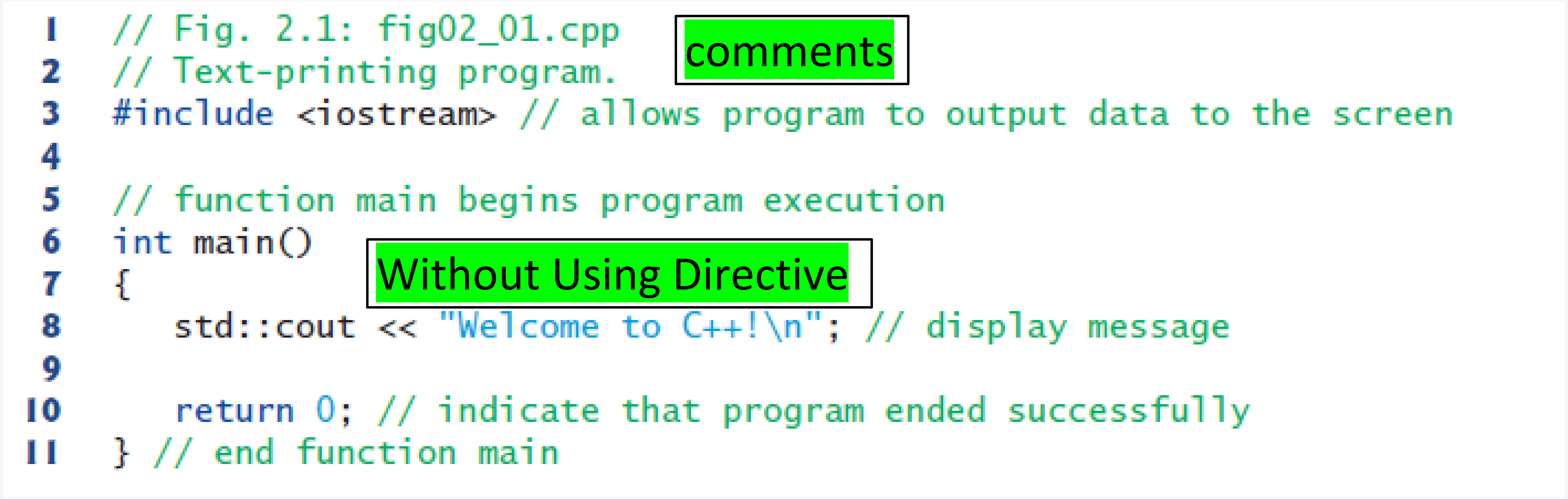
Preprocessor Directive

- A ***preprocessor directive*** is an instruction to the compiler.
- The preprocessor directive #include tells the compiler to insert another file into your source file.
- The type file usually included by #include is called a ***header file***.

C++ Application Build Process


Source Code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (cpp)
Include Header, Expand Macro (.i, .ii)
Compilation
Step 2: Compiler (gcc, g++)
Assembly Code (.s)
Assemble
Step 3: Assembler (as)
Machine Code (.o, .obj)
Static Library (.lib, .a)
Linking
Step 4: Linker (ld)
Executable Machine Code (.exe)

A Basic C++ Program



```cpp
// Fig. 2.1: fig02_01.cpp
// Text-printing program.
#include <iostream> // allows program to output data to the screen

// function main begins program execution
int main()
{
    std::cout << "Welcome to C++!\n"; // display message

    return 0; // indicate that program ended successfully
} // end function main
```

comments

Without Using Directive

Comments

```
// comments.cpp
// demonstrates comments
```

```
/* this is an old-style comment */
```

```
/* this
is a
potentially
very long
multiline
comment
*/
```

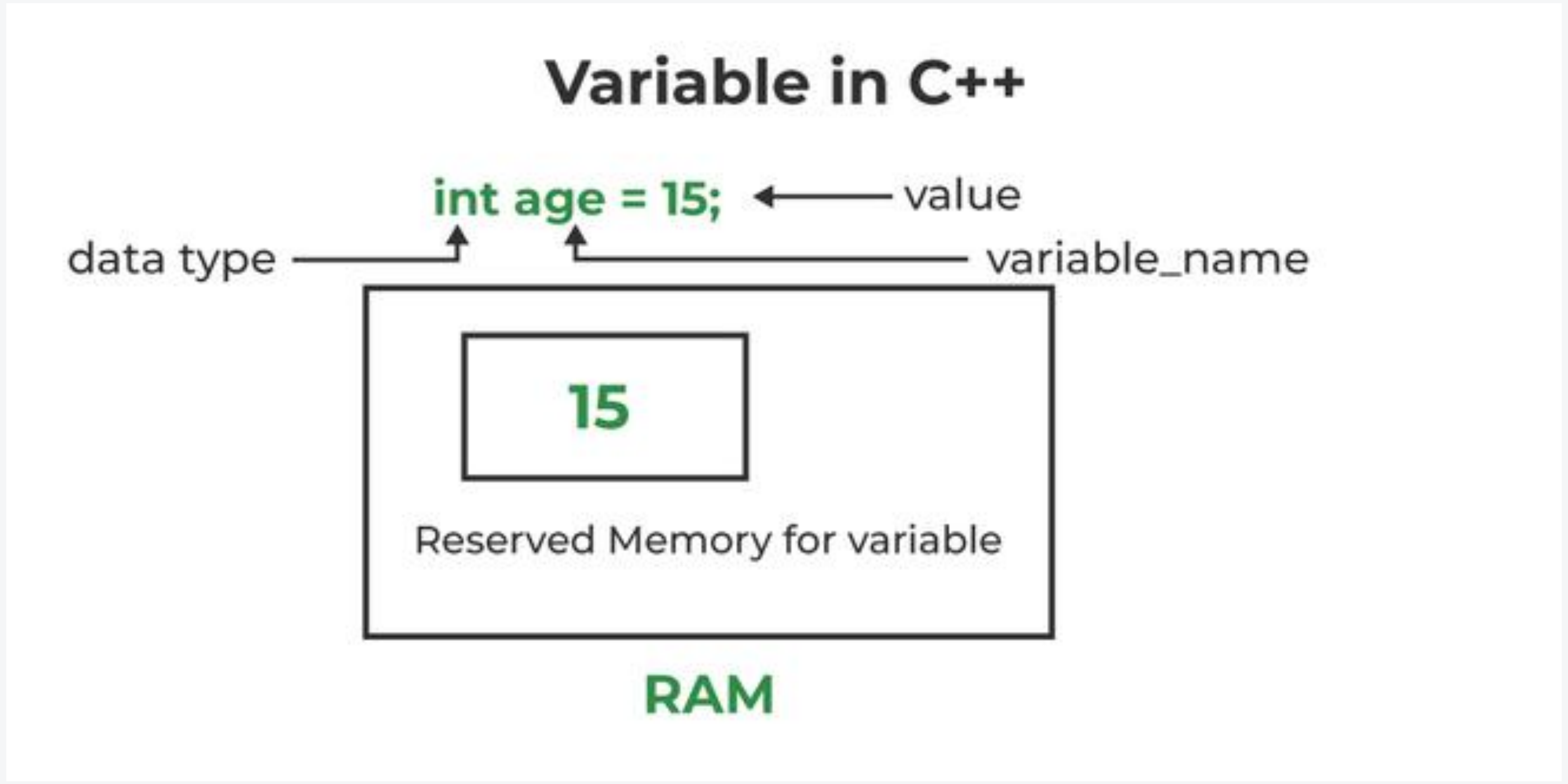
Whitespaces

- Whitespace is defined as spaces, returns, and tabs.
- These characters are invisible to the compiler.
- The first line of the program, starting with #include, is a preprocessor directive, which must be written on one line.
- Also, string constants, such as “Every age has a language of its own”, cannot be broken into separate lines. (If you need a long string constant, you can insert a backslash (\) at the line break or divide the string into two separate strings, each surrounded by quotes.)

04.

Variables

Variables



Variable Name/Identifier

- Numbers, alphabets, and _ allowed
- First character should be letter or underscore
- Case sensitive
- Do not use keywords

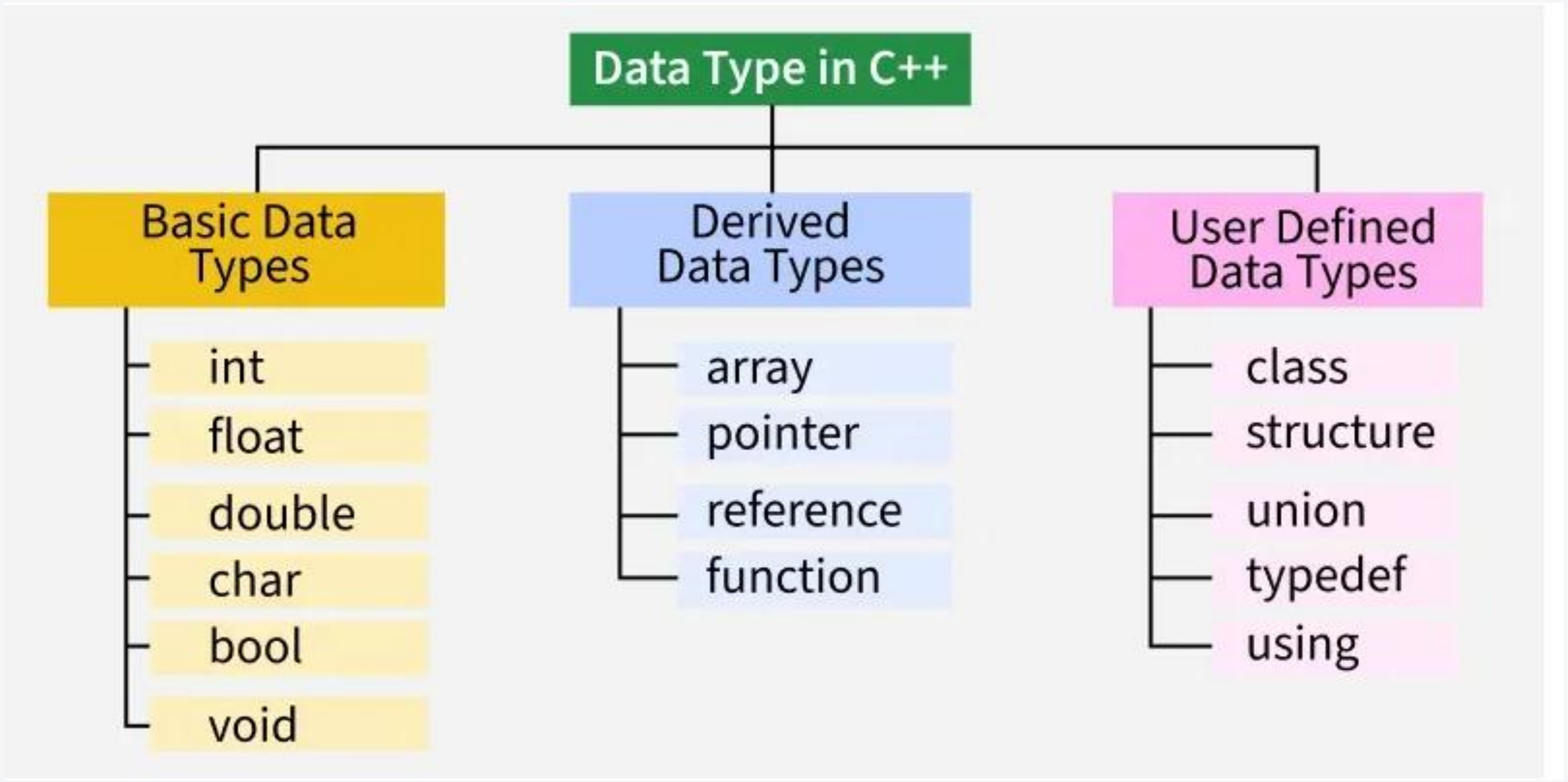
Example Code

Variable declaration

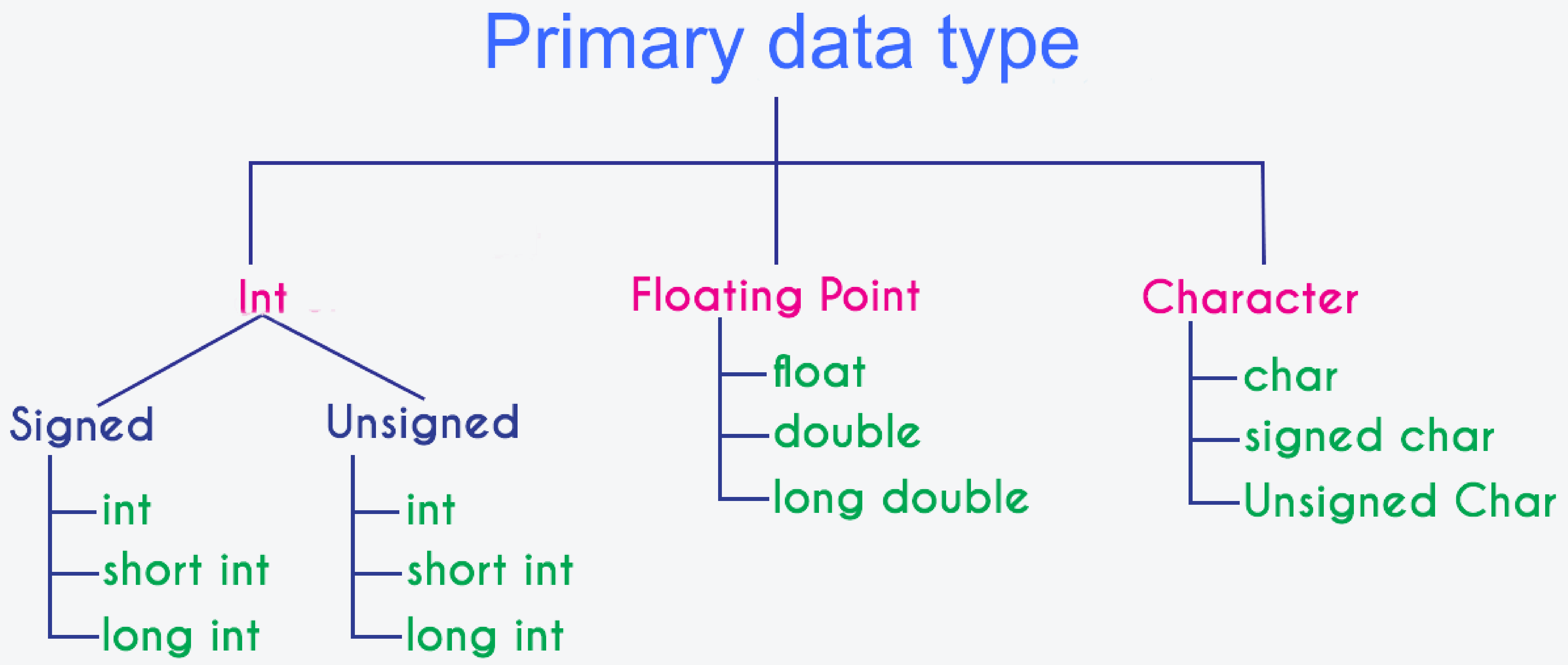
Variable definition

```
// intvars.cpp
// demonstrates integer variables
#include <iostream>
using namespace std;

int main()
   {
   int var1;                   //define var1
   int var2;                   //define var2

   var1 = 20;                  //assign value to var1
   var2 = var1 + 10;           //assign value to var2
   cout << “var1+10 is ”;      //output text
   cout << var2 << endl;       //output value of var2
   return 0;
   }
```

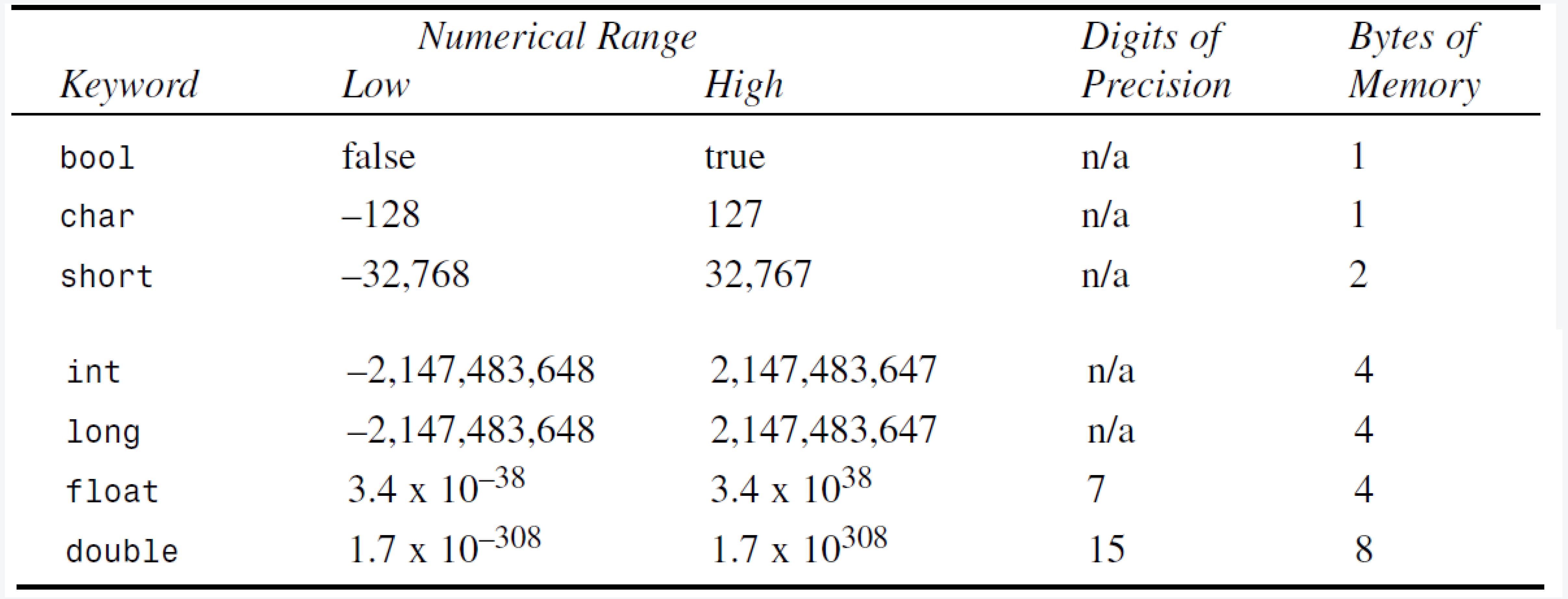
Data Types


Data Type in C++
Basic Data Types
int
float
double
char
bool
void
Derived Data Types
array
pointer
reference
function
User Defined Data Types
class
structure
union
typedef
using

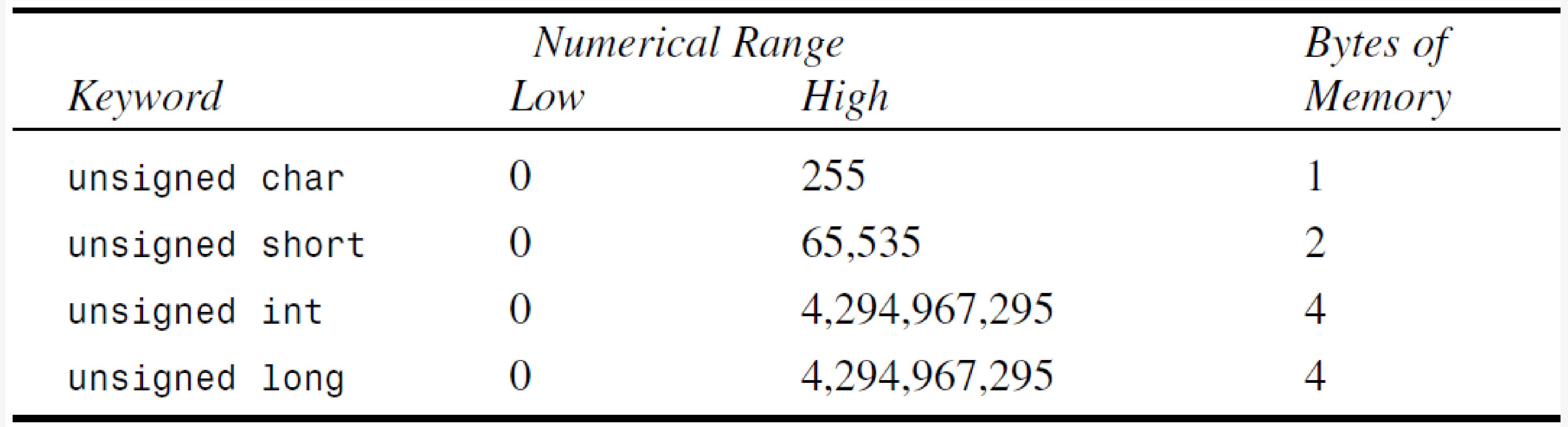
Data Types


Primary data type
Int
Floating Point
Character
Signed
Unsigned
int
short int
long int
int
short int
long int
float
double
long double
char
signed char
Unsigned Char

Data Types

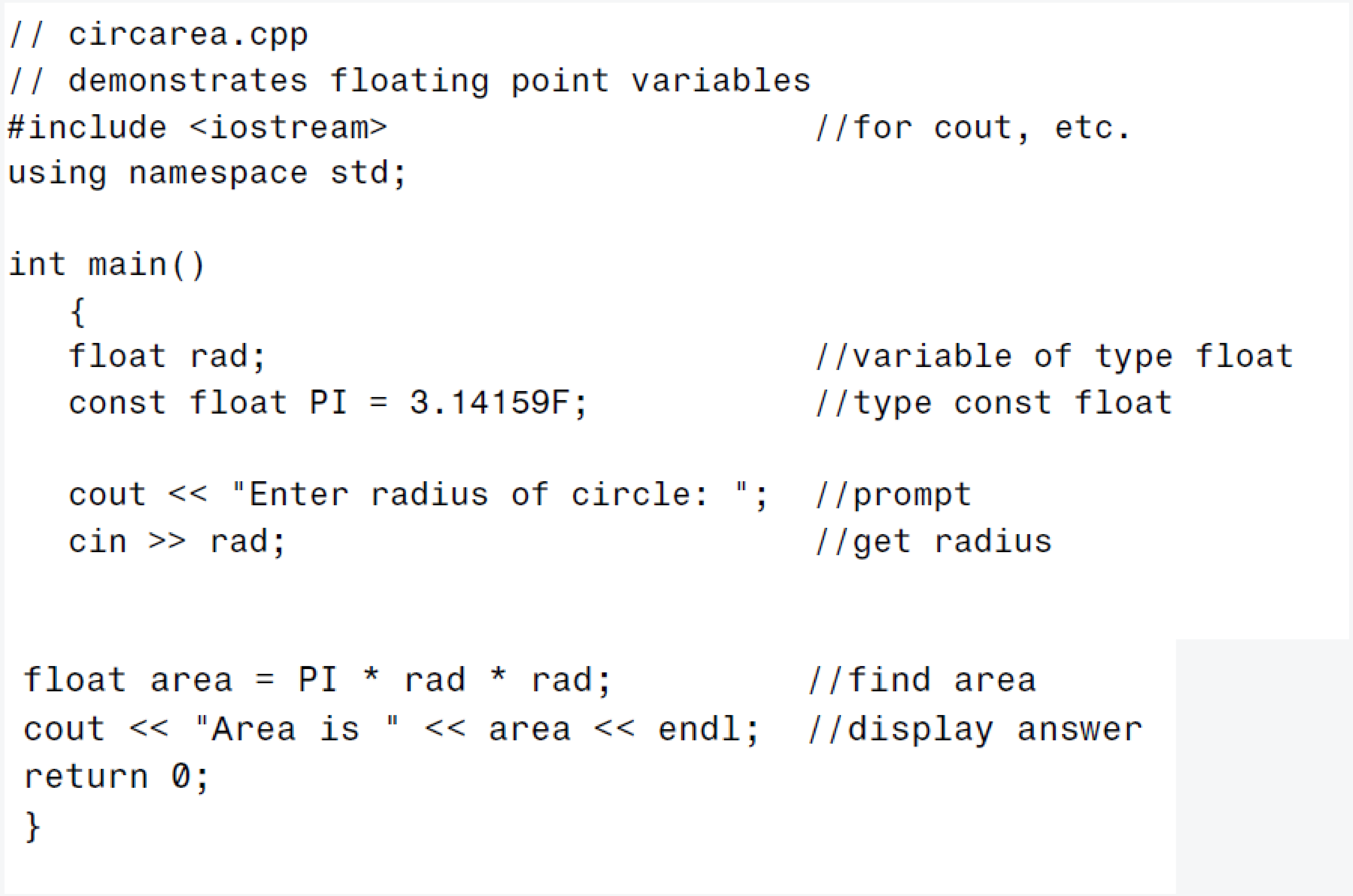


Keyword	*Numerical Range* *Low*	*High*	*Digits of Precision*	*Bytes of Memory*
`bool`	false	true	n/a	1
`char`	–128	127	n/a	1
`short`	–32,768	32,767	n/a	2
`int`	–2,147,483,648	2,147,483,647	n/a	4
`long`	–2,147,483,648	2,147,483,647	n/a	4
`float`	3.4×10^{-38}	3.4×10^{38}	7	4
`double`	1.7×10^{-308}	1.7×10^{308}	15	8



Keyword	*Numerical Range* *Low*	*High*	*Bytes of Memory*
`unsigned char`	0	255	1
`unsigned short`	0	65,535	2
`unsigned int`	0	4,294,967,295	4
`unsigned long`	0	4,294,967,295	4

Example Code



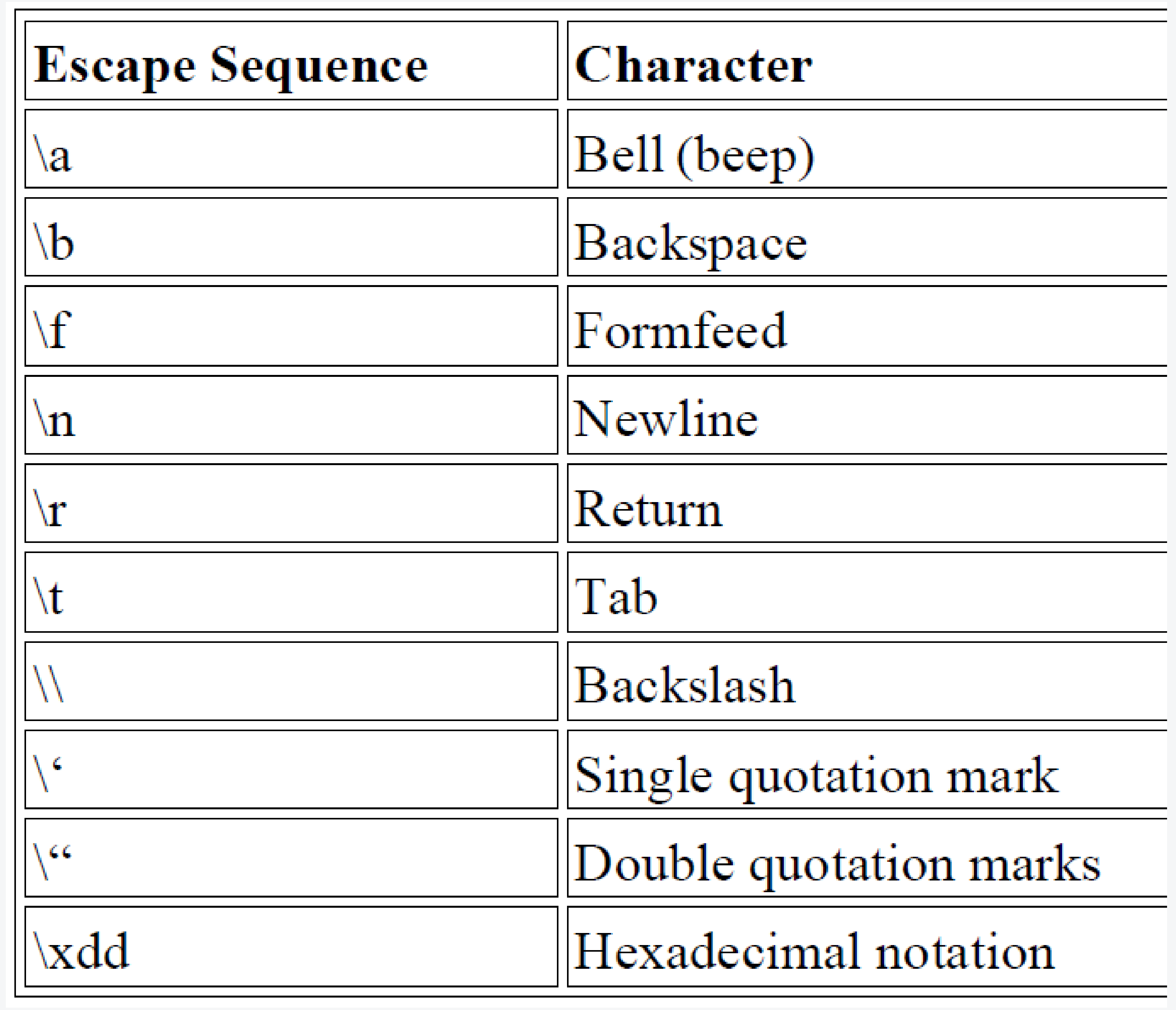
```
// circarea.cpp
// demonstrates floating point variables
#include <iostream>                       //for cout, etc.
using namespace std;

int main()
   {
   float rad;                             //variable of type float
   const float PI = 3.14159F;             //type const float

   cout << "Enter radius of circle: ";    //prompt
   cin >> rad;                            //get radius


 float area = PI * rad * rad;            //find area
 cout << "Area is " << area << endl;     //display answer
 return 0;
 }
```

Example Code

```
// charvars.cpp
// demonstrates character variables
#include &ltiostream>          //for cout, etc.
using namespace std;

int main()
   {
   char charvar1 = 'A';     //define char variable as character
   char charvar2 = '\t';    //define char variable as tab

   cout << charvar1;        //display character
   cout << charvar2;        //display character
   charvar1 = 'B';          //set char variable to char constant
   cout << charvar1;        //display character
   cout << '\n';            //display newline character
   return 0;
   }
```

Escape Sequences

- Special way to write characters that can't be typed directly and have special meaning.
- It starts with a backslash (\).
- They are used inside strings or characters.



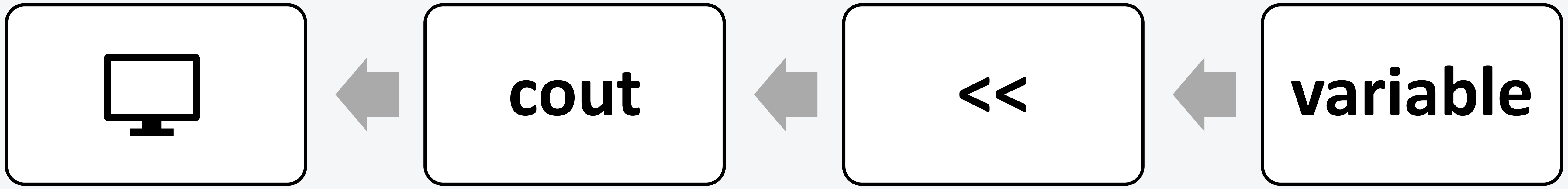
Escape Sequence	Character
\a	Bell (beep)
\b	Backspace
\f	Formfeed
\n	Newline
\r	Return
\t	Tab
\\	Backslash
\‘	Single quotation mark
\“	Double quotation marks
\xdd	Hexadecimal notation

05.

Input & Output

Output using cout

```
    cout << "var1+10 is ";   //output text
    cout << var2 << endl;    //output value of var2
```



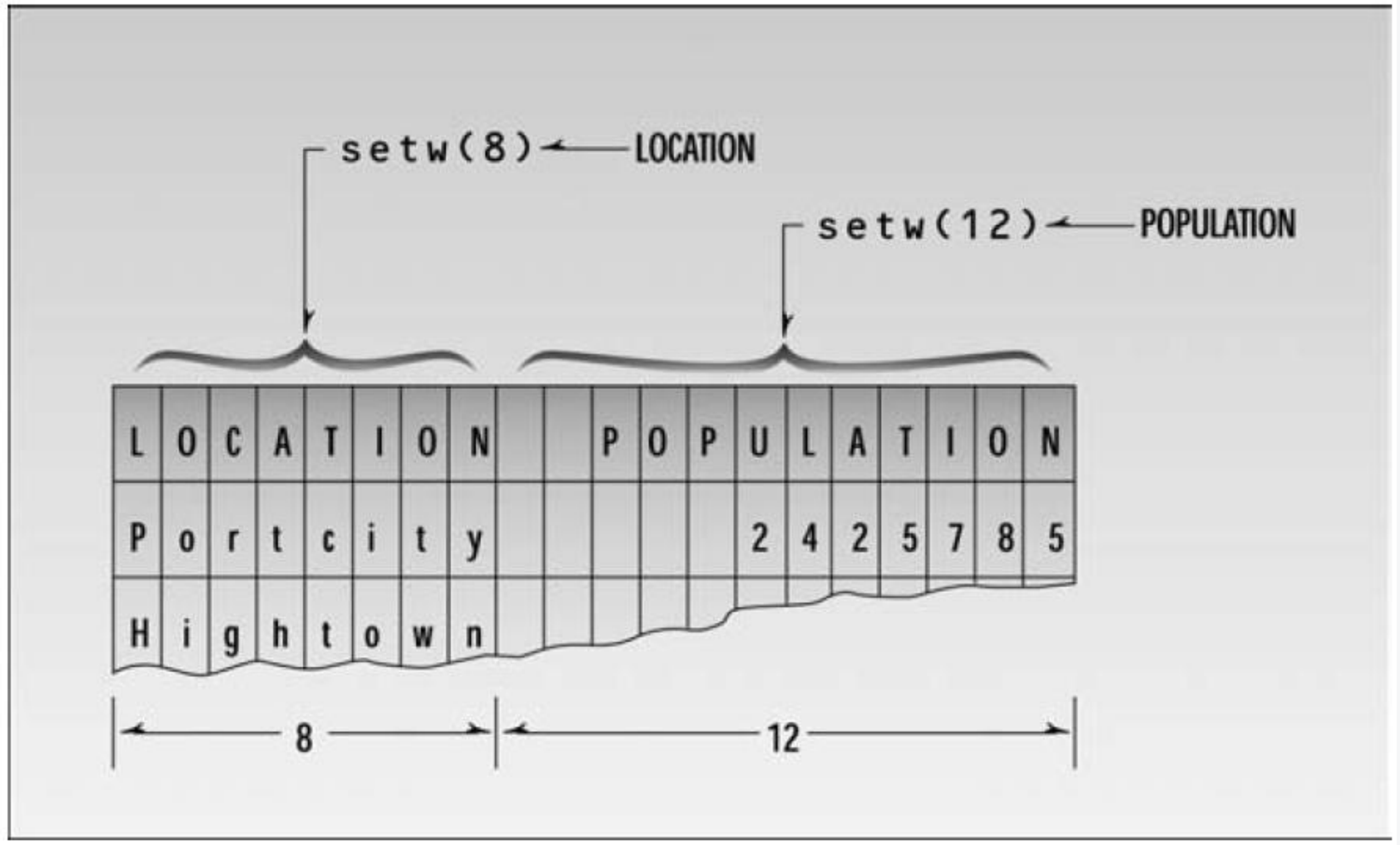
Insertion / put to operator

Setw manipulator

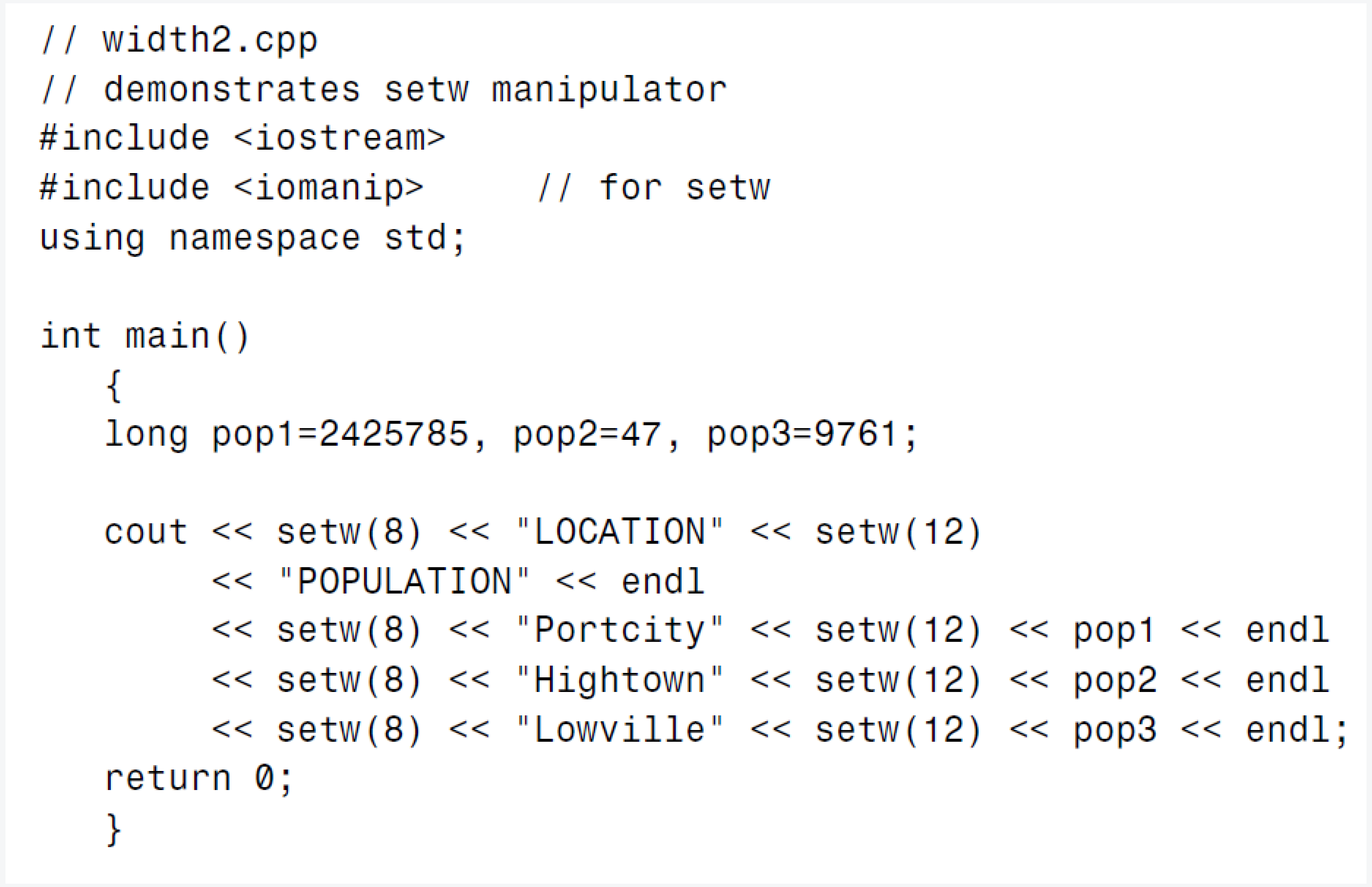
```
// width1.cpp
// demonstrates need for setw manipulator
#include <iostream>
using namespace std;

int main()
   {
   long pop1=2425785, pop2=47, pop3=9761;

   cout << "LOCATION " << "POP." << endl
        << "Portcity " << pop1 << endl
        << "Hightown " << pop2 << endl
        << "Lowville " << pop3 << endl;
   return 0;
   }
```

Here’s the output from this program:

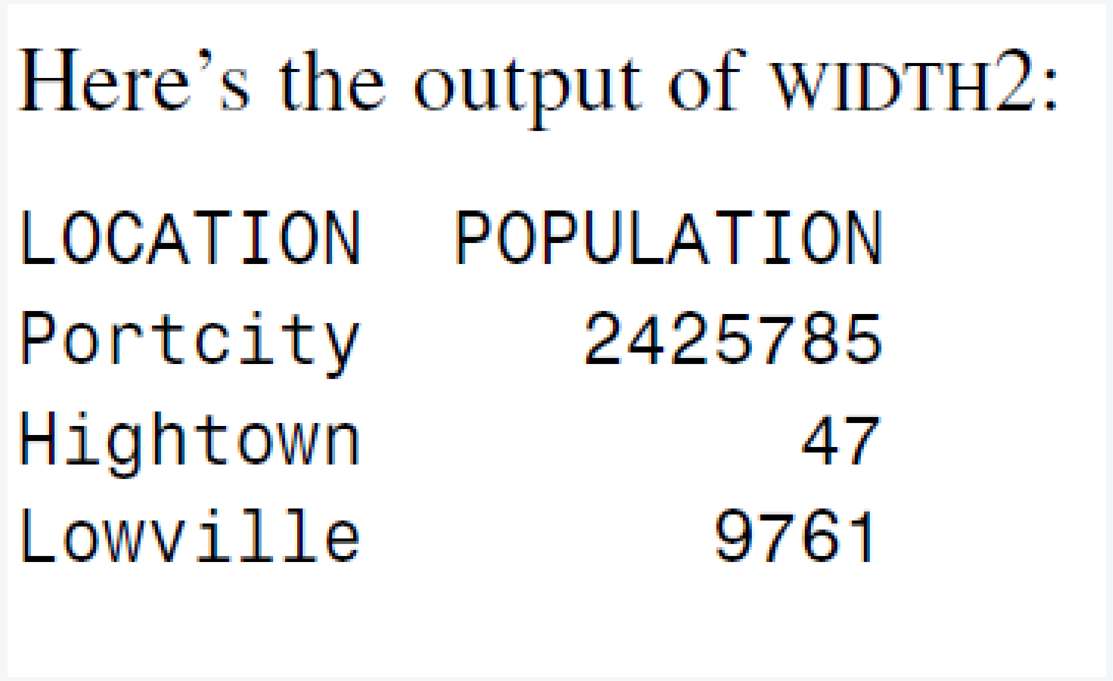
```
LOCATION POP.
Portcity 2425785
Hightown 47
Lowville 9761
```



Setw manipulator



```cpp
// width2.cpp
// demonstrates setw manipulator
#include <iostream>
#include <iomanip>     // for setw
using namespace std;

int main()
   {
   long pop1=2425785, pop2=47, pop3=9761;

   cout << setw(8) << "LOCATION" << setw(12)
        << "POPULATION" << endl
        << setw(8) << "Portcity" << setw(12) << pop1 << endl
        << setw(8) << "Hightown" << setw(12) << pop2 << endl
        << setw(8) << "Lowville" << setw(12) << pop3 << endl;
   return 0;
   }
```



Here’s the output of WIDTH2:

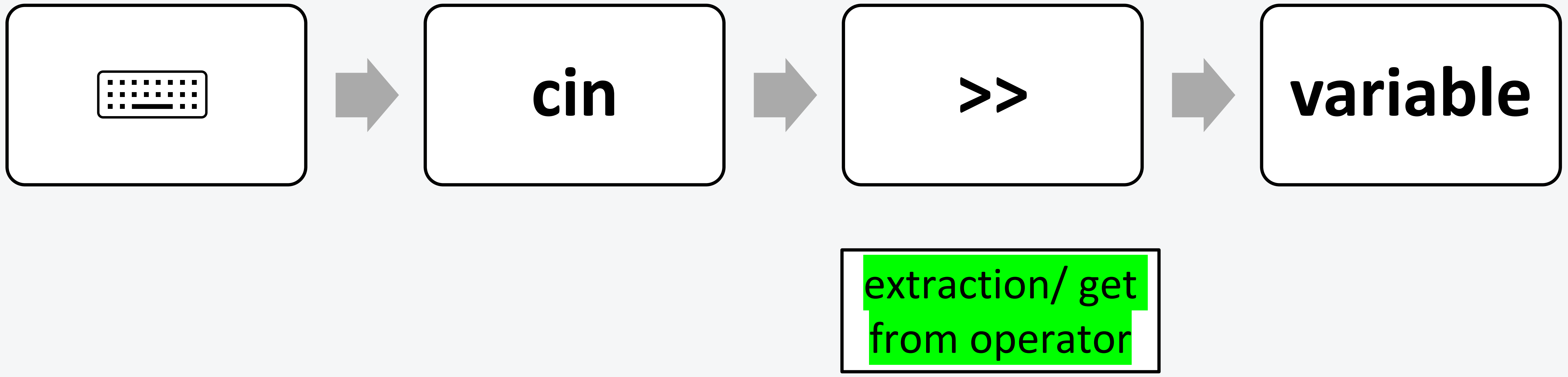
```
LOCATION  POPULATION
Portcity     2425785
Hightown          47
Lowville        9761
```

Input using cin

```
// fahren.cpp
// demonstrates cin, newline
#include <iostream>
using namespace std;

int main()
   {
   int ftemp;  //for temperature in fahrenheit

   cout << "Enter temperature in fahrenheit: ";
   cin >> ftemp;
   int ctemp = (ftemp-32) * 5 / 9;
   cout << "Equivalent in Celsius is: " << ctemp << '\n';
   return 0;
   }
```

input

Input using cin



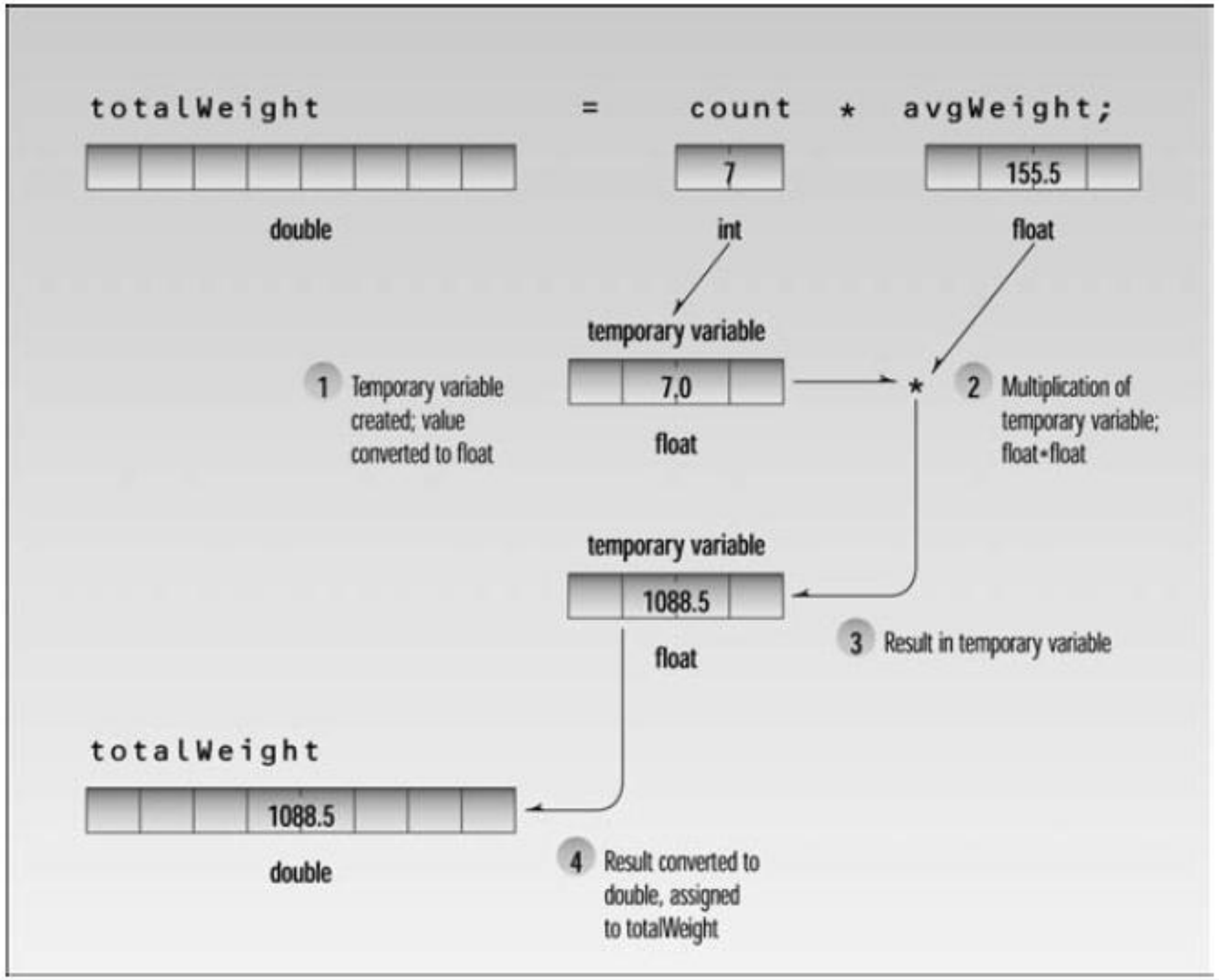
06.

Type Conversion

Automatic Type Conversion

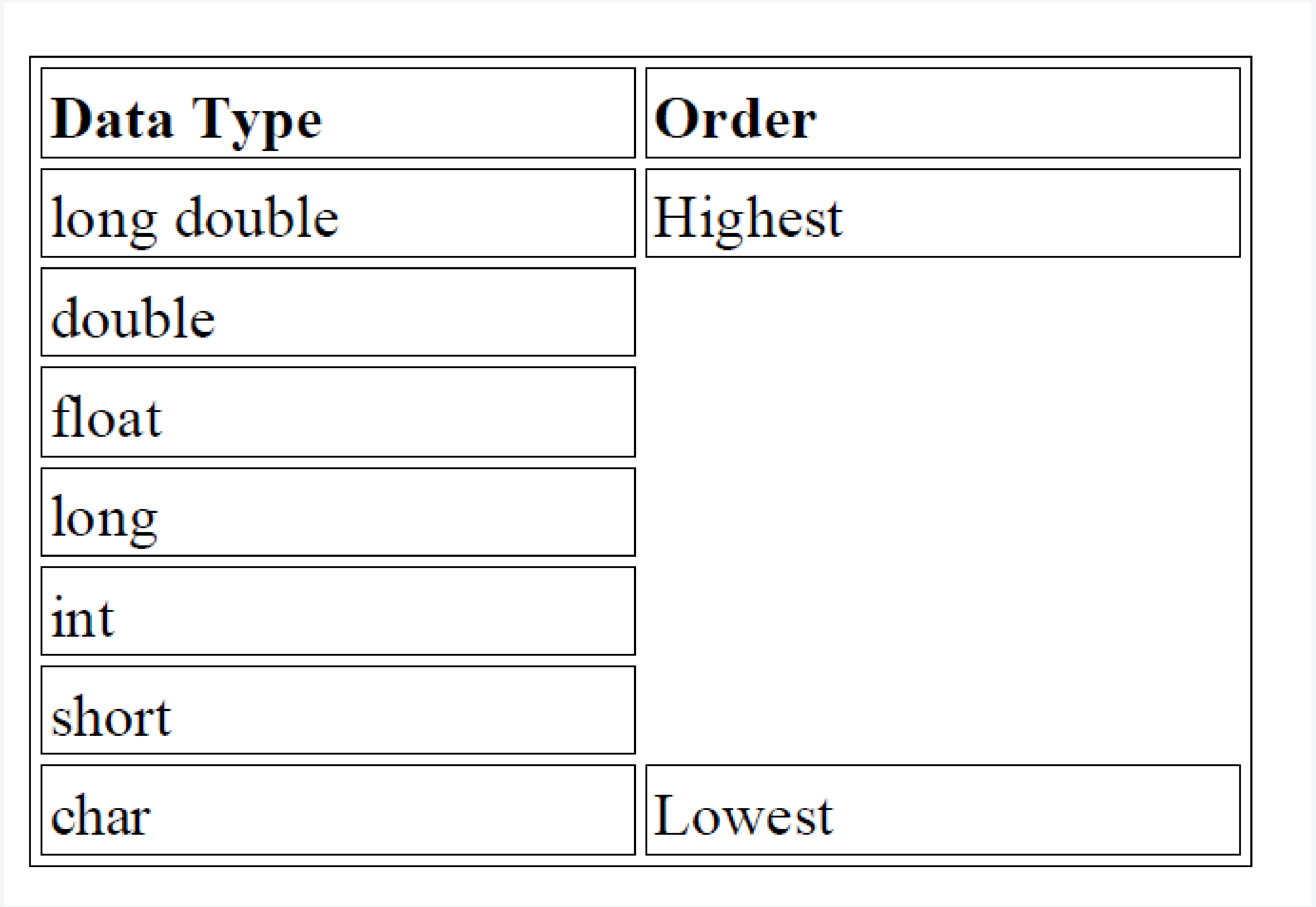
```
int main()
    {
    int count = 7;
    float avgWeight = 155.5F;

    double totalWeight = count * avgWeight;
    cout << "totalWeight=" << totalWeight << endl;
    return 0;
    }
```

Automatic Type Conversion


totalWeight = count * avgWeight;
7
155.5
double
int
float
temporary variable
7.0
float
1 Temporary variable created; value converted to float
2 Multiplication of temporary variable; float*float
temporary variable
1088.5
float
3 Result in temporary variable
totalWeight
1088.5
double
4 Result converted to double, assigned to totalWeight

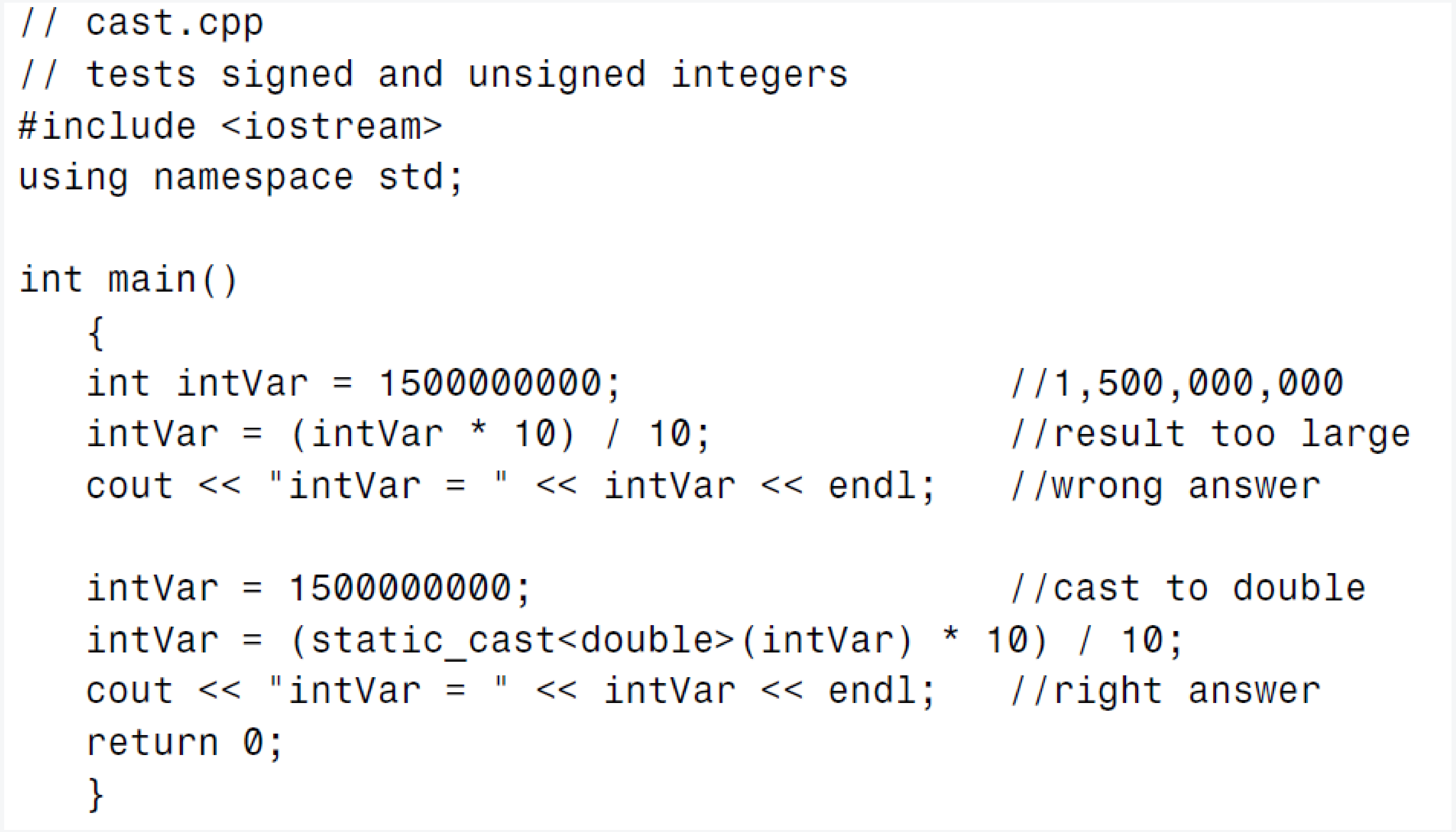
Automatic Type Conversion

- The arithmetic operators such as + and * like to operate on two operands of the same type.
- When two operands of different types are encountered in the same expression, the lower-type variable is converted to the type of the higher-type variable.



Data Type	**Order**
long double	Highest
double	
float	
long	
int	
short	
char	Lowest

Type Casting

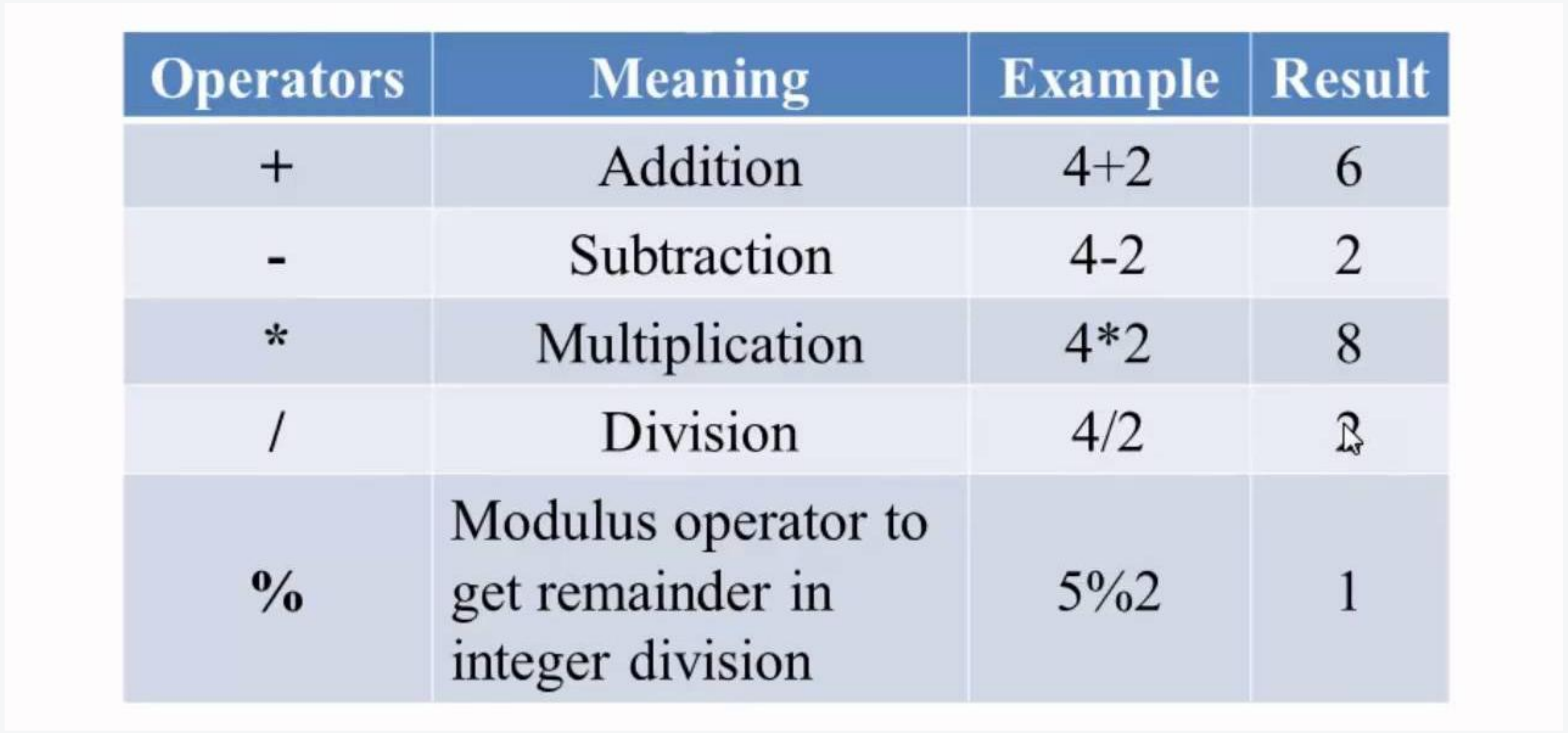


```
// cast.cpp
// tests signed and unsigned integers
#include <iostream>
using namespace std;

int main()
   {
   int intVar = 1500000000;                    //1,500,000,000
   intVar = (intVar * 10) / 10;                //result too large
   cout << "intVar = " << intVar << endl;      //wrong answer

   intVar = 1500000000;                        //cast to double
   intVar = (static_cast<double>(intVar) * 10) / 10;
   cout << "intVar = " << intVar << endl;      //right answer
   return 0;
   }
```

07.

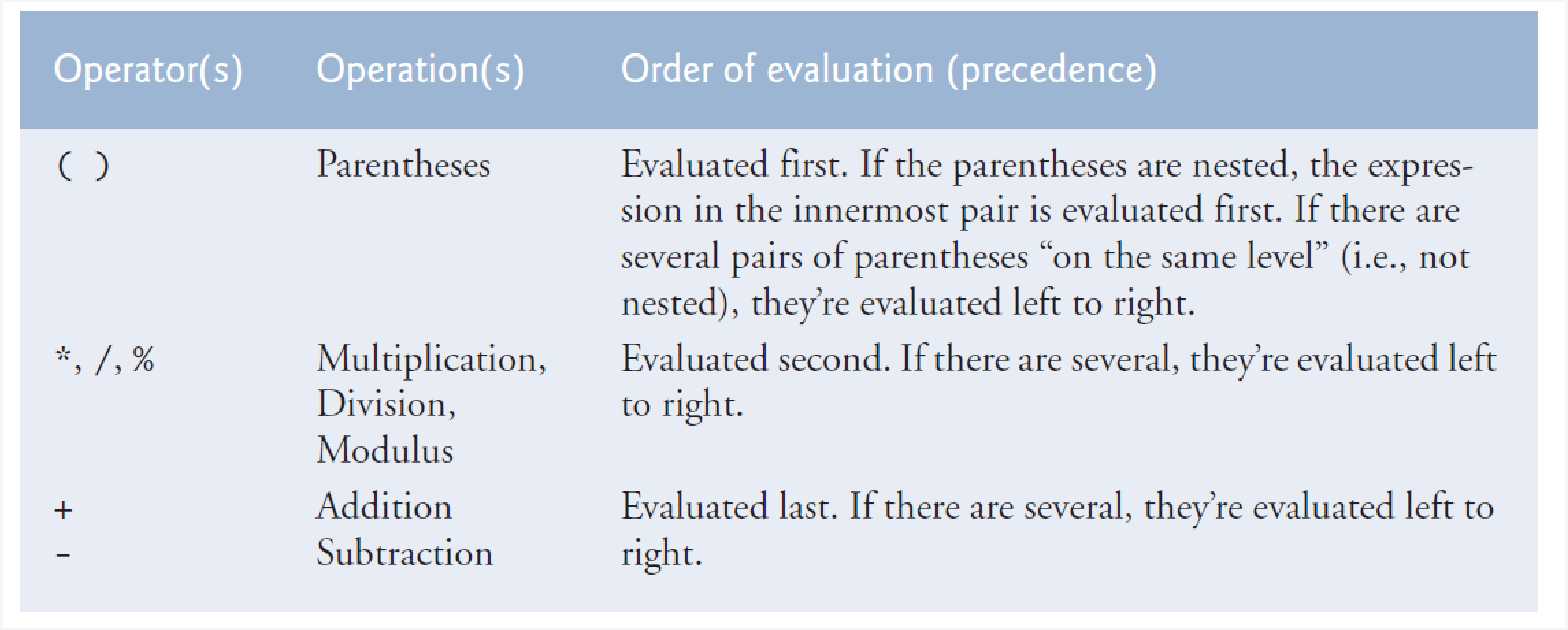
Arithmetic Oper.

Arithmetic Operations



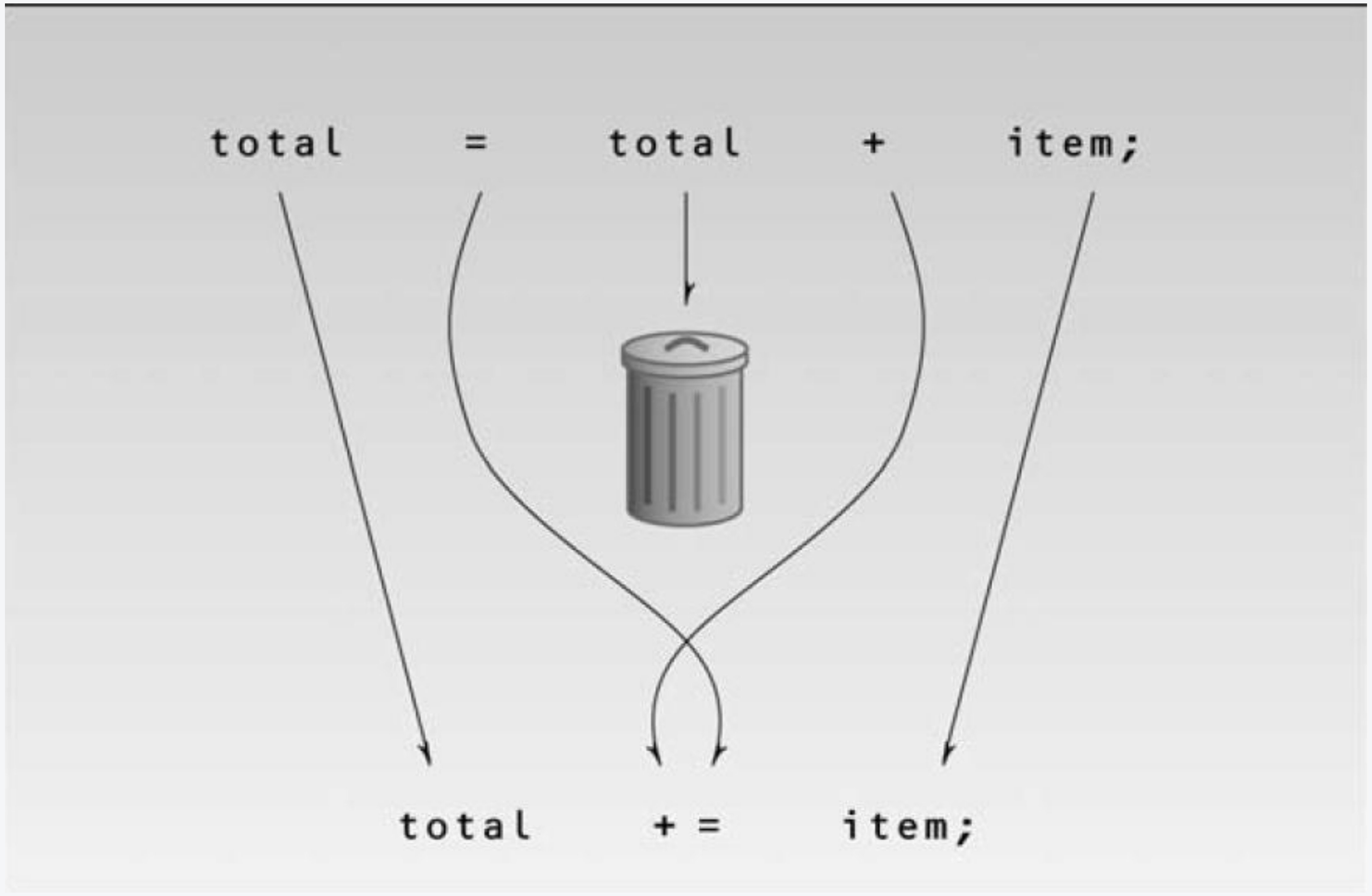
Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1

Arithmetic Operations Precedence

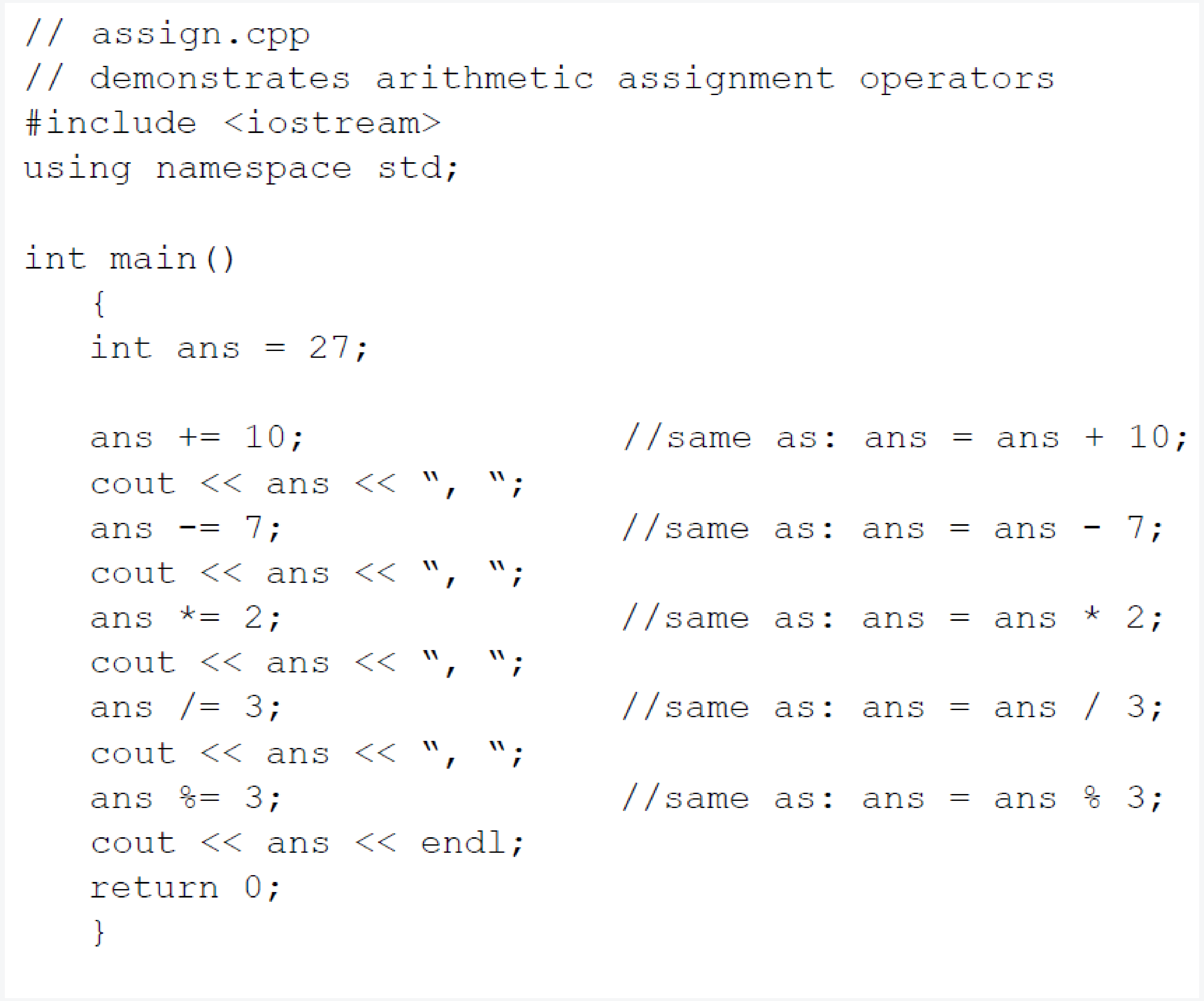


Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
*, /, %	Multiplication, Division, Modulus	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

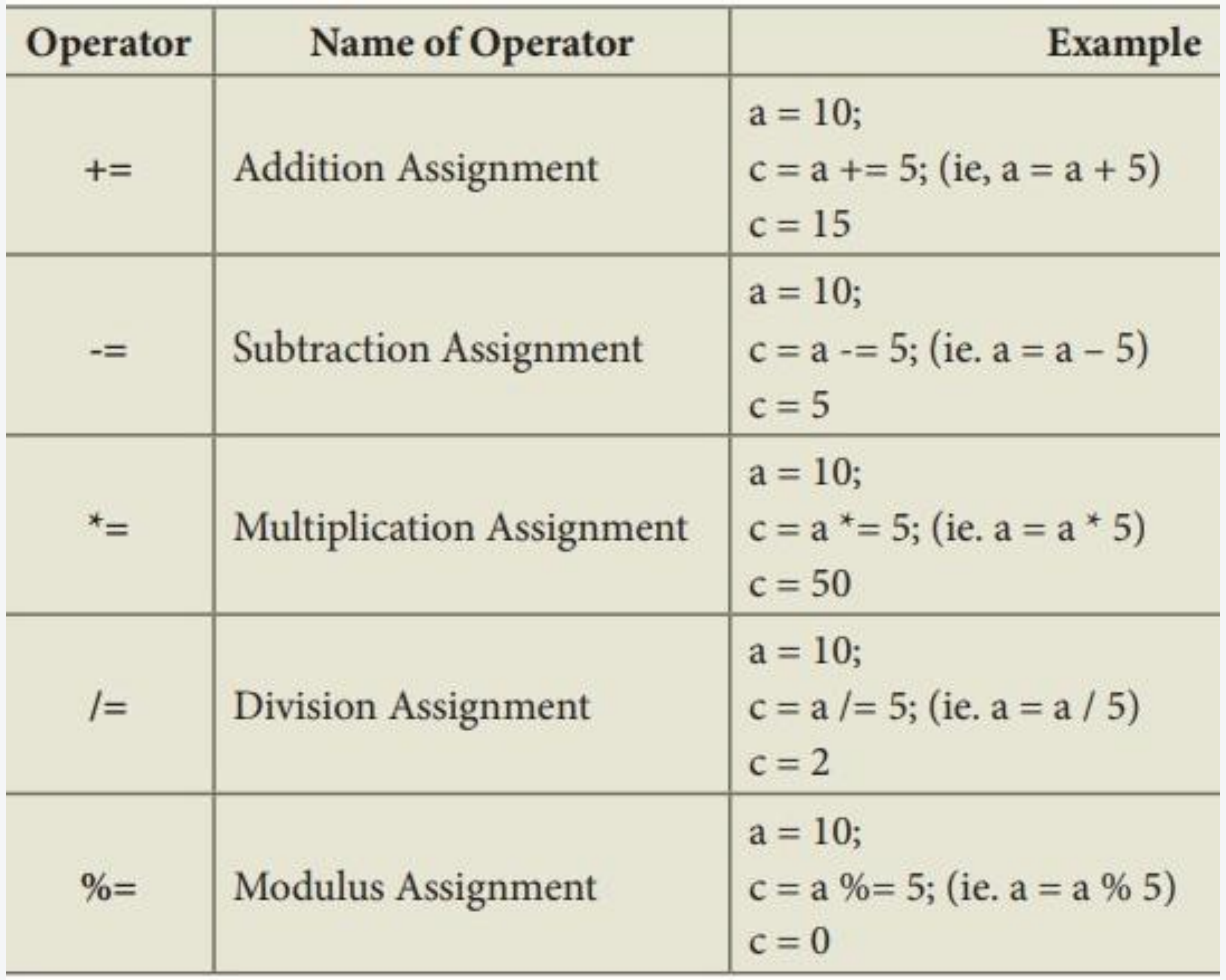
Arithmetic Assignment Operations



Arithmetic Assignment Operations



```
// assign.cpp
// demonstrates arithmetic assignment operators
#include <iostream>
using namespace std;

int main()
   {
   int ans = 27;

   ans += 10;              //same as: ans = ans + 10;
   cout << ans << “, “;
   ans -= 7;               //same as: ans = ans - 7;
   cout << ans << “, “;
   ans *= 2;               //same as: ans = ans * 2;
   cout << ans << “, “;
   ans /= 3;               //same as: ans = ans / 3;
   cout << ans << “, “;
   ans %= 3;               //same as: ans = ans % 3;
   cout << ans << endl;
   return 0;
   }
```



Operator	Name of Operator	Example

+=	Addition Assignment	a = 10;
		c = a += 5; (ie, a = a + 5)
		c = 15

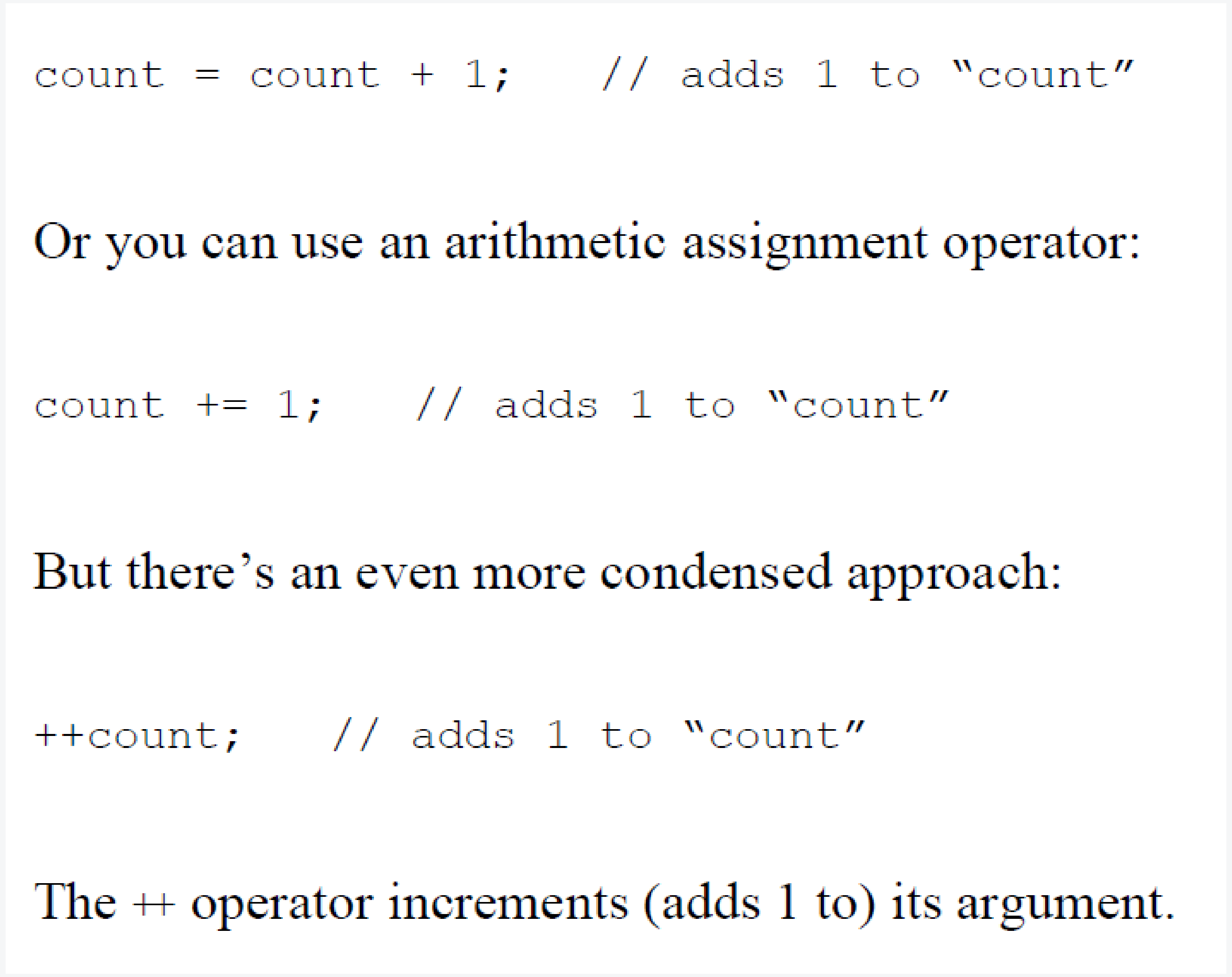
-=	Subtraction Assignment	a = 10;
		c = a -= 5; (ie. a = a – 5)
		c = 5

*=	Multiplication Assignment	a = 10;
		c = a *= 5; (ie. a = a * 5)
		c = 50

/=	Division Assignment	a = 10;
		c = a /= 5; (ie. a = a / 5)
		c = 2

%=	Modulus Assignment	a = 10;
		c = a %= 5; (ie. a = a % 5)
		c = 0

Increment Operator



```
count = count + 1;    // adds 1 to "count"
```

Or you can use an arithmetic assignment operator:

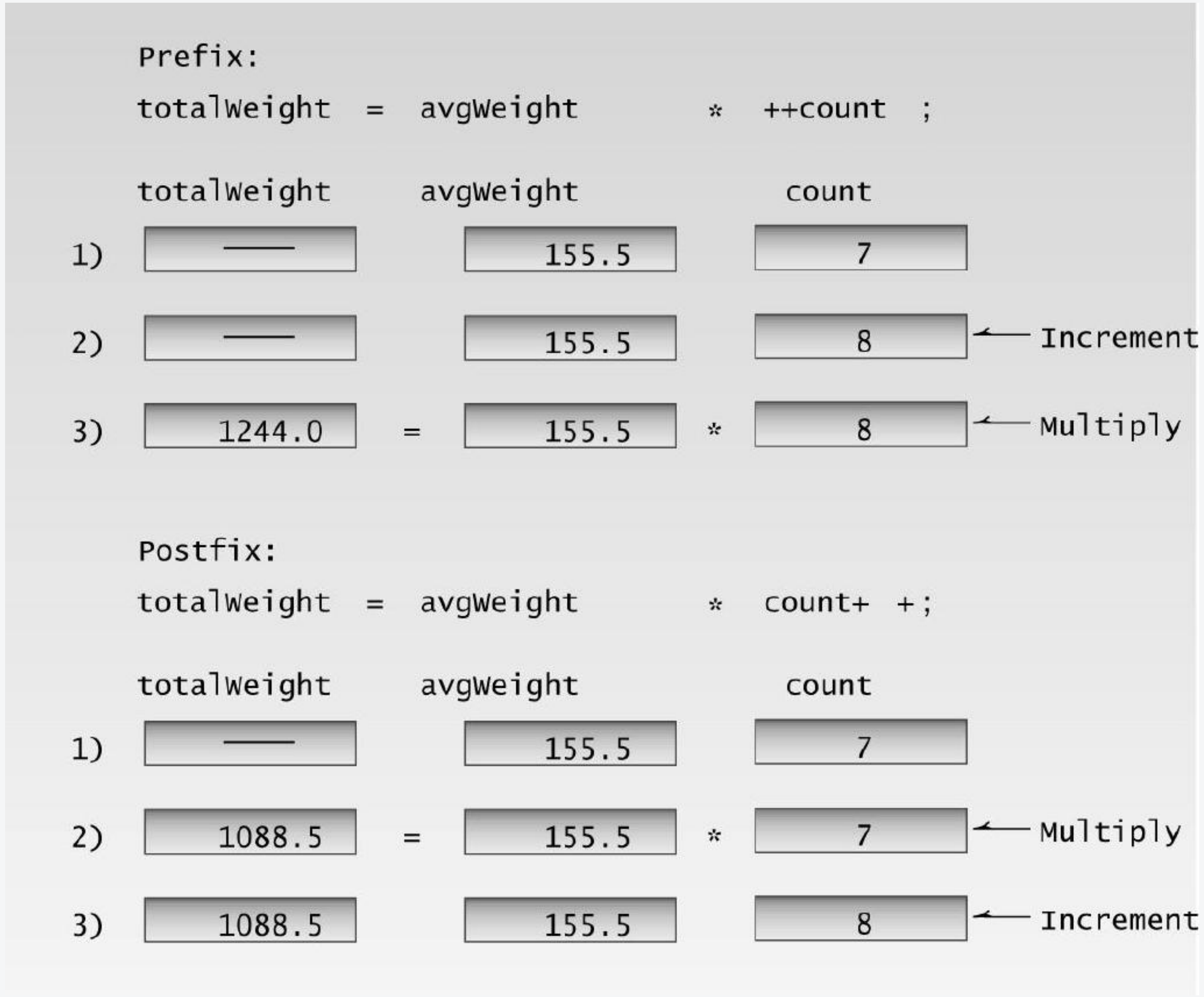
```
count += 1;    // adds 1 to "count"
```

But there's an even more condensed approach:

```
++count;    // adds 1 to "count"
```

The ++ operator increments (adds 1 to) its argument.

Postfix and Prefix



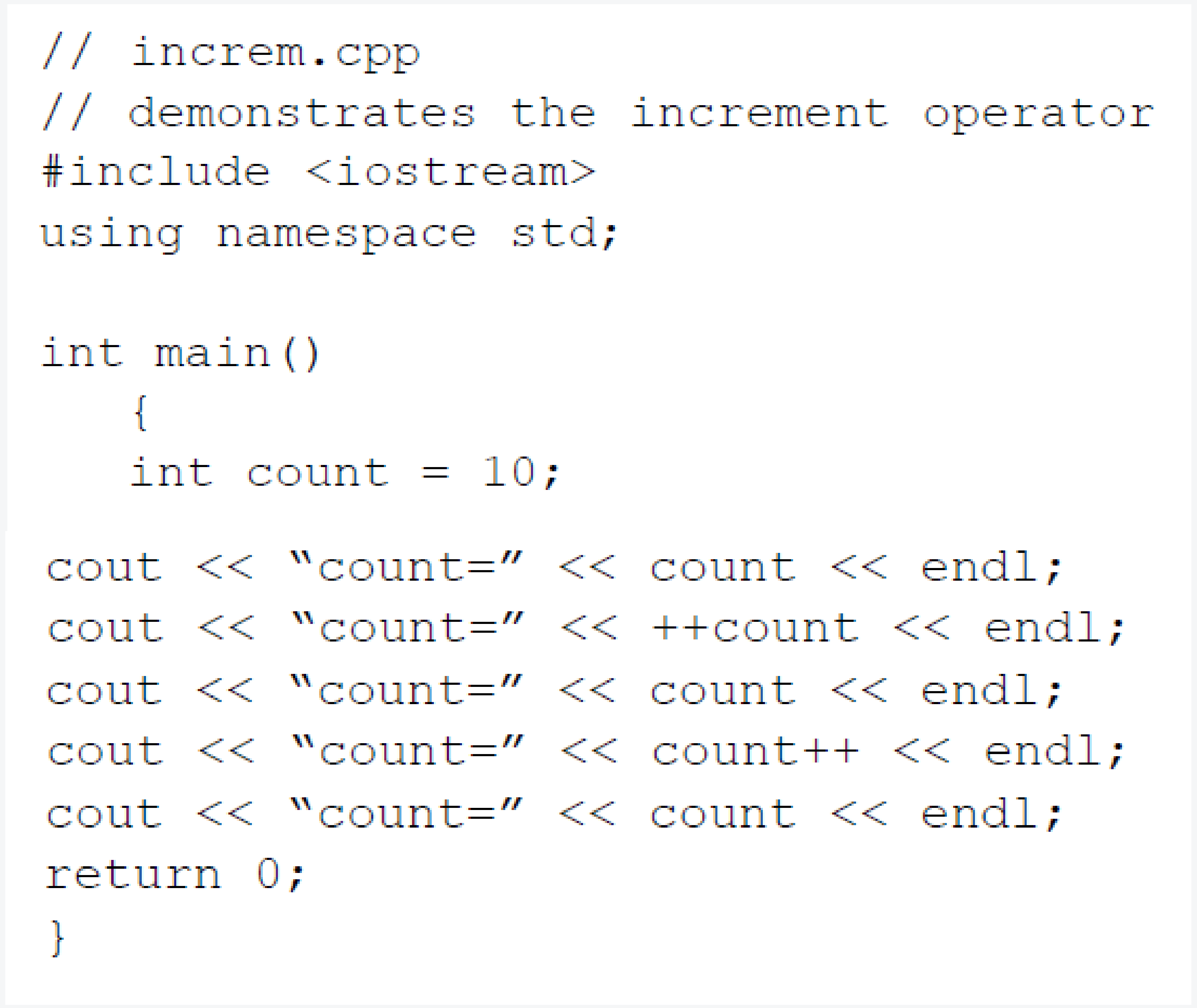
```
Prefix:
totalWeight  =  avgWeight     *  ++count  ;
```

totalWeight avgWeight count
1) — 155.5 7
2) — 155.5 8 Increment
3) 1244.0 = 155.5 * 8 Multiply

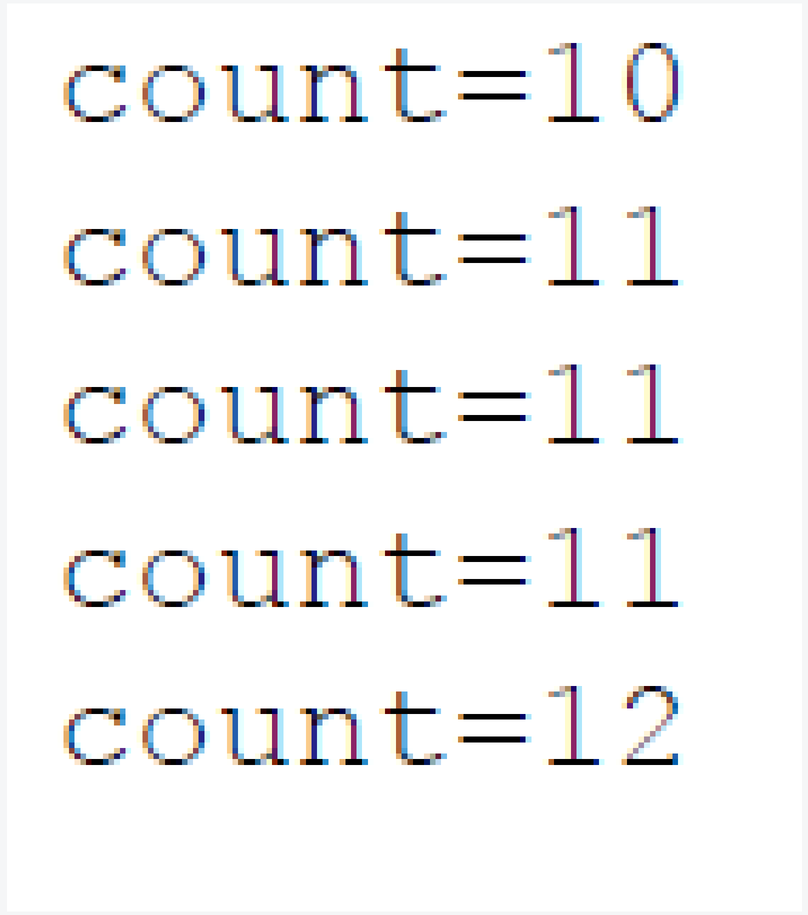
```
Postfix:
totalWeight  =  avgWeight     *  count+ +;
```

totalWeight avgWeight count
1) — 155.5 7
2) 1088.5 = 155.5 * 7 Multiply
3) 1088.5 155.5 8 Increment

Increment Operator



```
// increm.cpp
// demonstrates the increment operator
#include <iostream>
using namespace std;

int main()
   {
   int count = 10;
cout << “count=” << count << endl;
cout << “count=” << ++count << endl;
cout << “count=” << count << endl;
cout << “count=” << count++ << endl;
cout << “count=” << count << endl;
return 0;
}
```



```
count=10
count=11
count=11
count=11
count=12
```

08.

Tasks

Task-1

- You can convert temperature from degrees Celsius to degrees Fahrenheit by multiplying by 9/5 and adding 32.
- Write a program that allows the user to enter a floating-point number representing degrees Celsius, and then displays the corresponding degrees Fahrenheit.

Task-2

- Write a program that generates the following output:

```
10
20
19
```

- Use an integer constant for the 10, an arithmetic assignment operator to generate the 20, and a decrement operator to generate the 19.

Thank You!