Variables and Constants

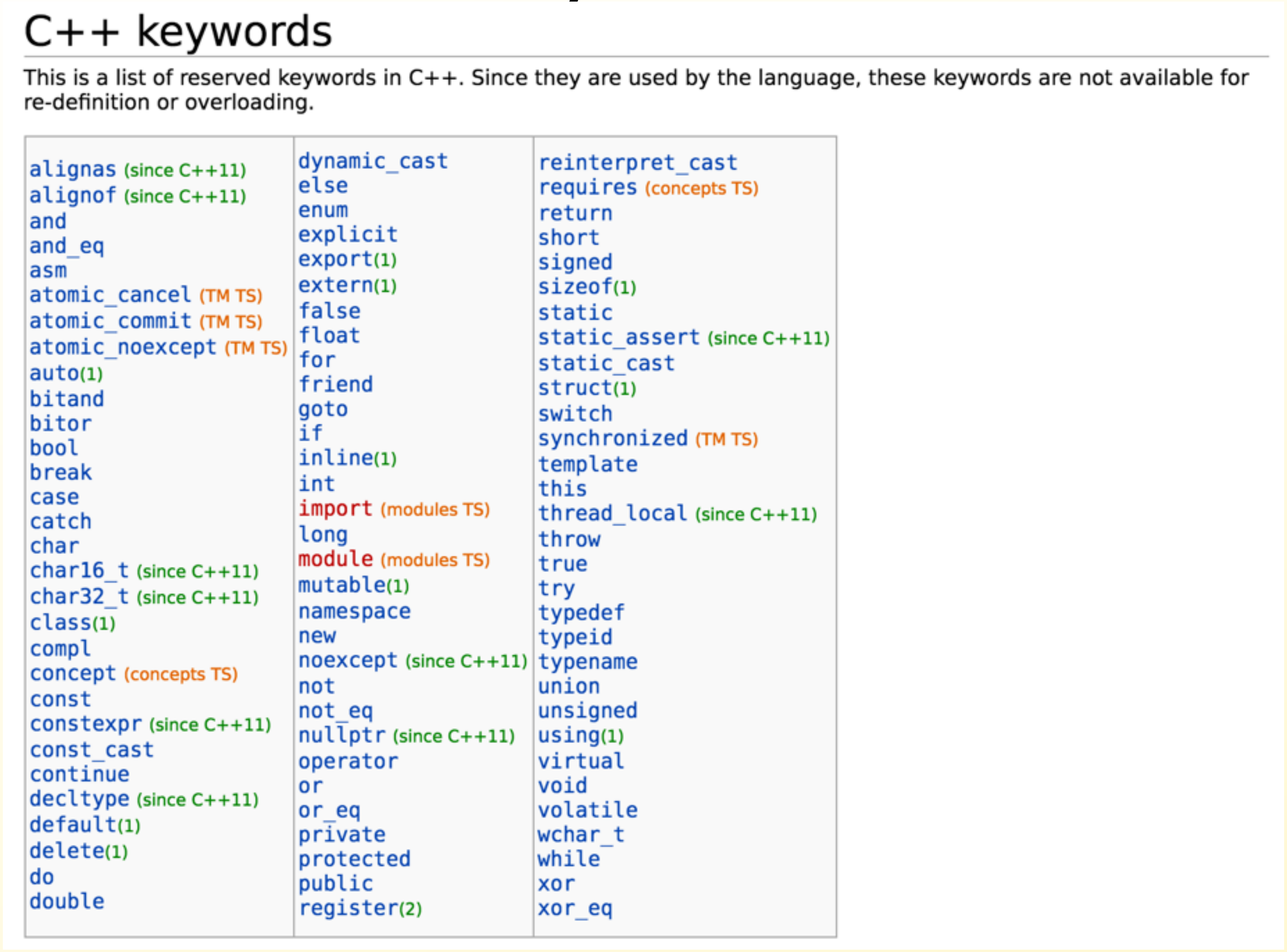
William Oquendo, woquendo@gmail.com

Credits: Computational Physics - Landau and Paez, Wikipedia, www.cplusplus.com/doc/tutorial , http://en.cppreference.com/w/cpp/language/, Stroupsup "The C++ programming language"

Identifiers

- A valid identifier is a sequence of one or more letters, digits, or underscore characters (_). Spaces, punctuation marks, and symbols cannot be part of an identifier. In addition, identifiers shall always begin with a letter. They can also begin with an underline character (_), but such identifiers are -on most cases- considered reserved for compiler-specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere. In no case can they begin with a digit.

Keywords



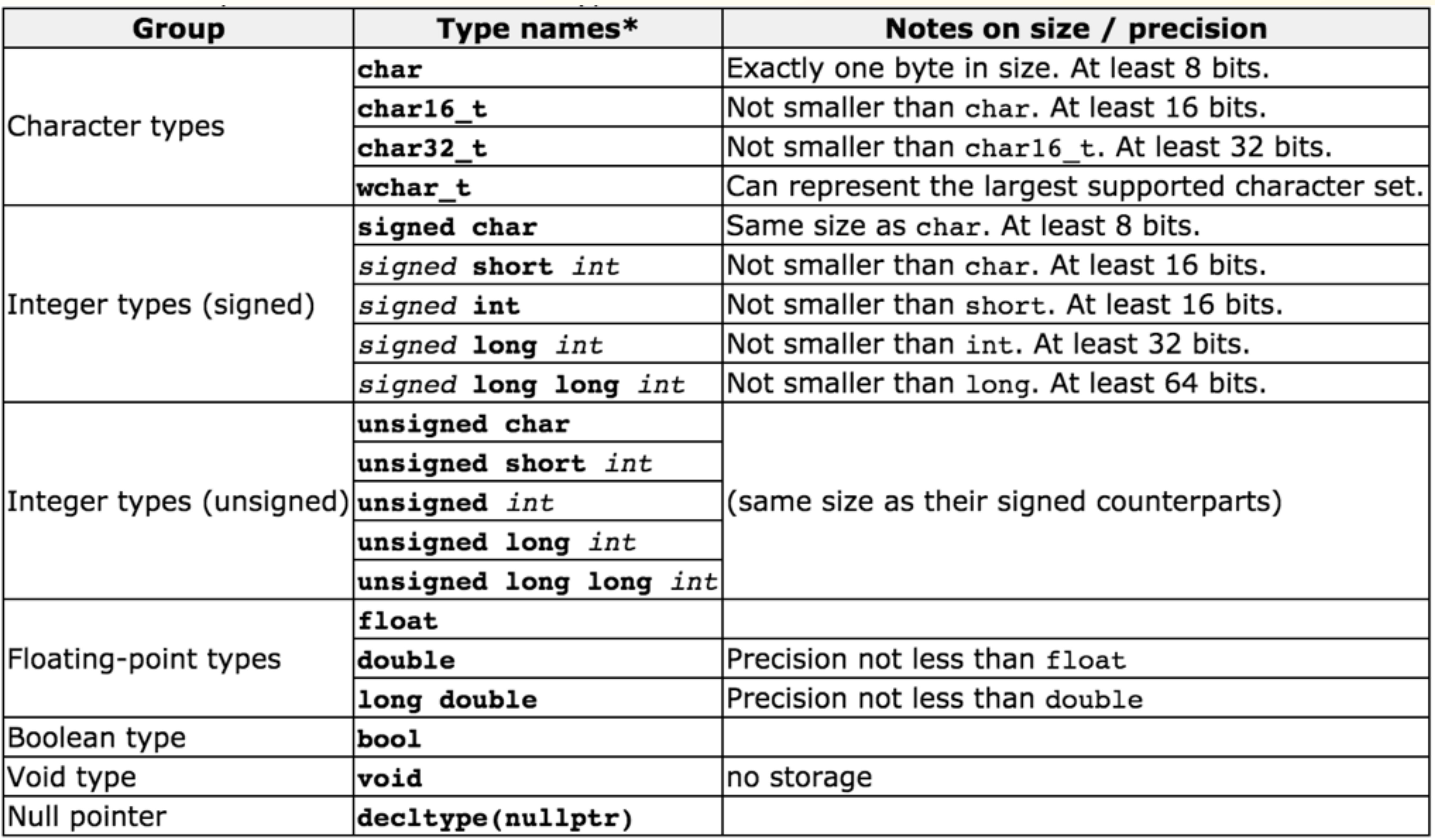
C++ keywords

This is a list of reserved keywords in C++. Since they are used by the language, these keywords are not available for re-definition or overloading.

alignas (since C++11)	dynamic_cast	reinterpret_cast
alignof (since C++11)	else	requires (concepts TS)
and	enum	return
and_eq	explicit	short
asm	export(1)	signed
atomic_cancel (TM TS)	extern(1)	sizeof(1)
atomic_commit (TM TS)	false	static
atomic_noexcept (TM TS)	float	static_assert (since C++11)
auto(1)	for	static_cast
bitand	friend	struct(1)
bitor	goto	switch
bool	if	synchronized (TM TS)
break	inline(1)	template
case	int	this
catch	import (modules TS)	thread_local (since C++11)
char	long	throw
char16_t (since C++11)	module (modules TS)	true
char32_t (since C++11)	mutable(1)	try
class(1)	namespace	typedef
compl	new	typeid
concept (concepts TS)	noexcept (since C++11)	typename
const	not	union
constexpr (since C++11)	not_eq	unsigned
const_cast	nullptr (since C++11)	using(1)
continue	operator	virtual
decltype (since C++11)	or	void
default(1)	or_eq	volatile
delete(1)	private	wchar_t
do	protected	while
double	public	xor
	register(2)	xor_eq

Types

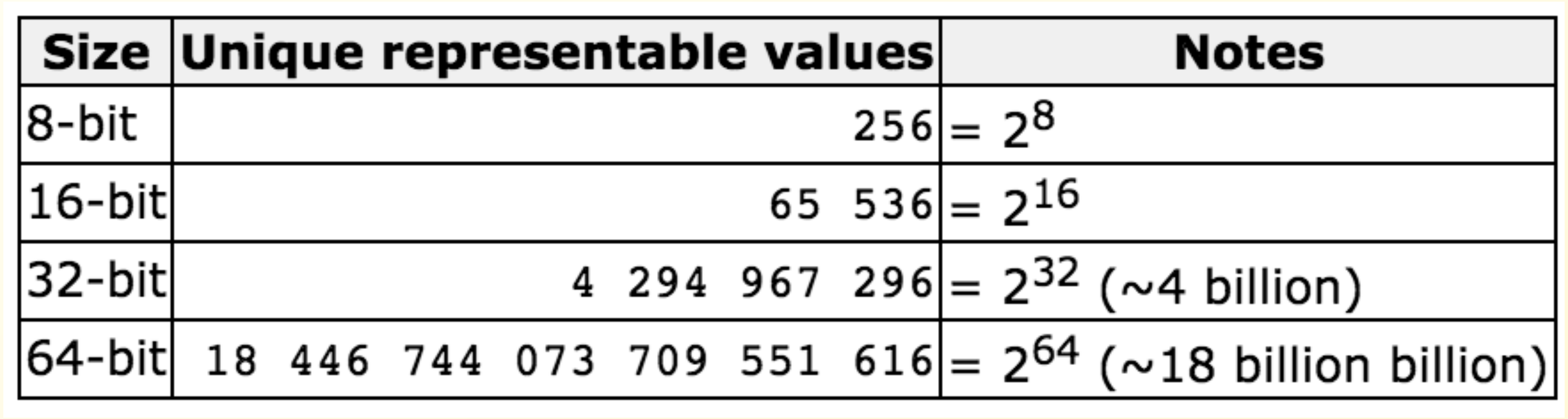
- Operator sizeof allows to know the size.



Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

Types

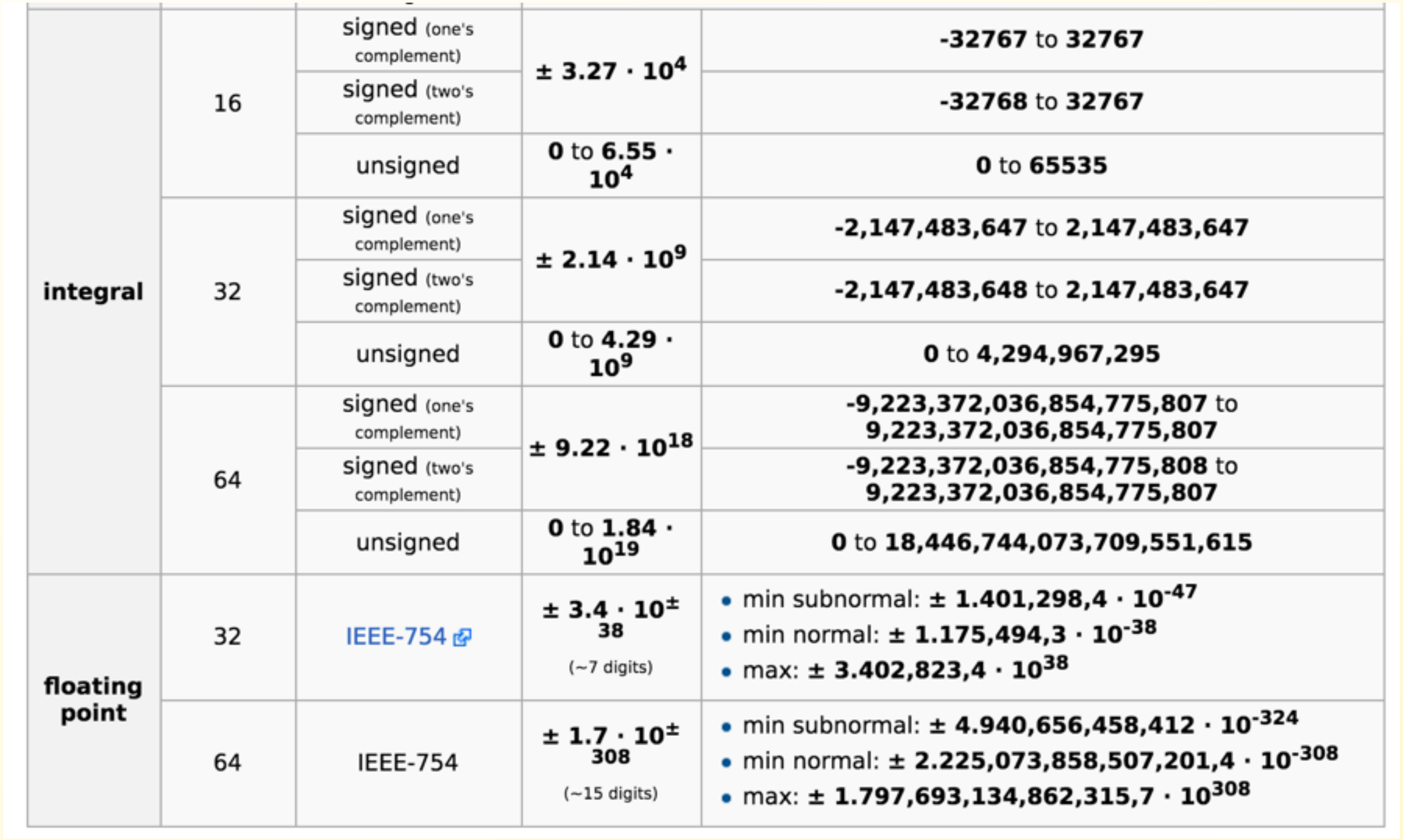
- Operator sizeof allows to know the size.



Size	Unique representable values	Notes
8-bit	256	$= 2^{8}$
16-bit	65 536	$= 2^{16}$
32-bit	4 294 967 296	$= 2^{32}$ (~4 billion)
64-bit	18 446 744 073 709 551 616	$= 2^{64}$ (~18 billion billion)

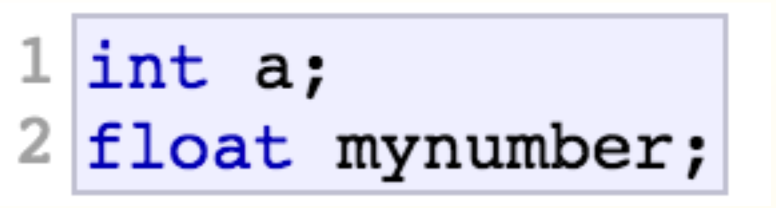
Special types and ranges

- INFINITY (isinf)
- NaN (Not a number) - (isnan)



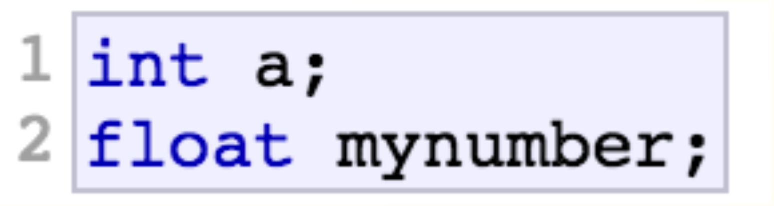
integral	16	signed (one's complement)	**± 3.27 · 10^4**	**-32767** to **32767**
		signed (two's complement)		**-32768** to **32767**
		unsigned	**0** to **6.55 · 10^4**	**0** to **65535**
	32	signed (one's complement)	**± 2.14 · 10^9**	**-2,147,483,647** to **2,147,483,647**
		signed (two's complement)		**-2,147,483,648** to **2,147,483,647**
		unsigned	**0** to **4.29 · 10^9**	**0** to **4,294,967,295**
	64	signed (one's complement)	**± 9.22 · 10^{18}**	**-9,223,372,036,854,775,807** to **9,223,372,036,854,775,807**
		signed (two's complement)		**-9,223,372,036,854,775,808** to **9,223,372,036,854,775,807**
		unsigned	**0** to **1.84 · 10^{19}**	**0** to **18,446,744,073,709,551,615**
floating point	32	IEEE-754	**± 3.4 · $10^{\pm 38}$** (~7 digits)	• min subnormal: **± 1.401,298,4 · 10^{-47}** • min normal: **± 1.175,494,3 · 10^{-38}** • max: **± 3.402,823,4 · 10^{38}**
	64	IEEE-754	**± 1.7 · $10^{\pm 308}$** (~15 digits)	• min subnormal: **± 4.940,656,458,412 · 10^{-324}** • min normal: **± 2.225,073,858,507,201,4 · 10^{-308}** • max: **± 1.797,693,134,862,315,7 · 10^{308}**

Declaration

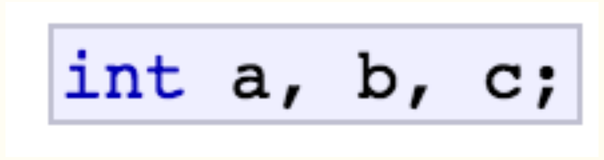


```
int a;
float mynumber;
```

Declaration



```
int a;
float mynumber;
```



```
int a, b, c;
```

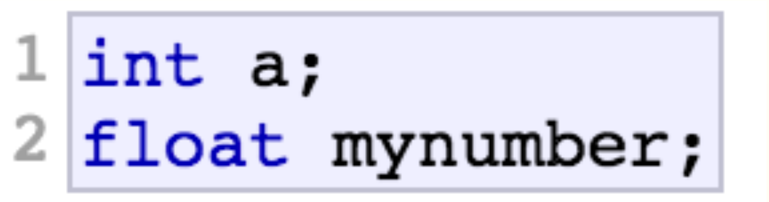
Declaration

```
int a;
float mynumber;
```

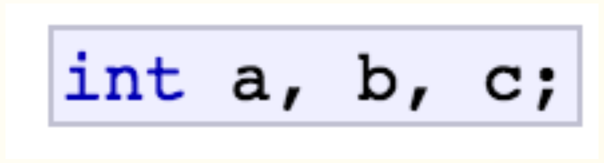
```
int a, b, c;
```

```
int a;
int b;
int c;
```

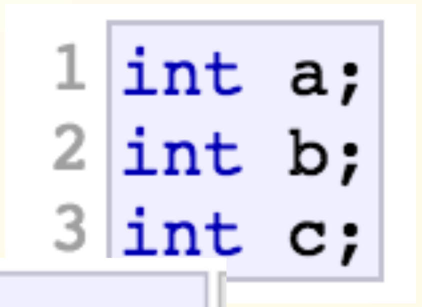
Declaration



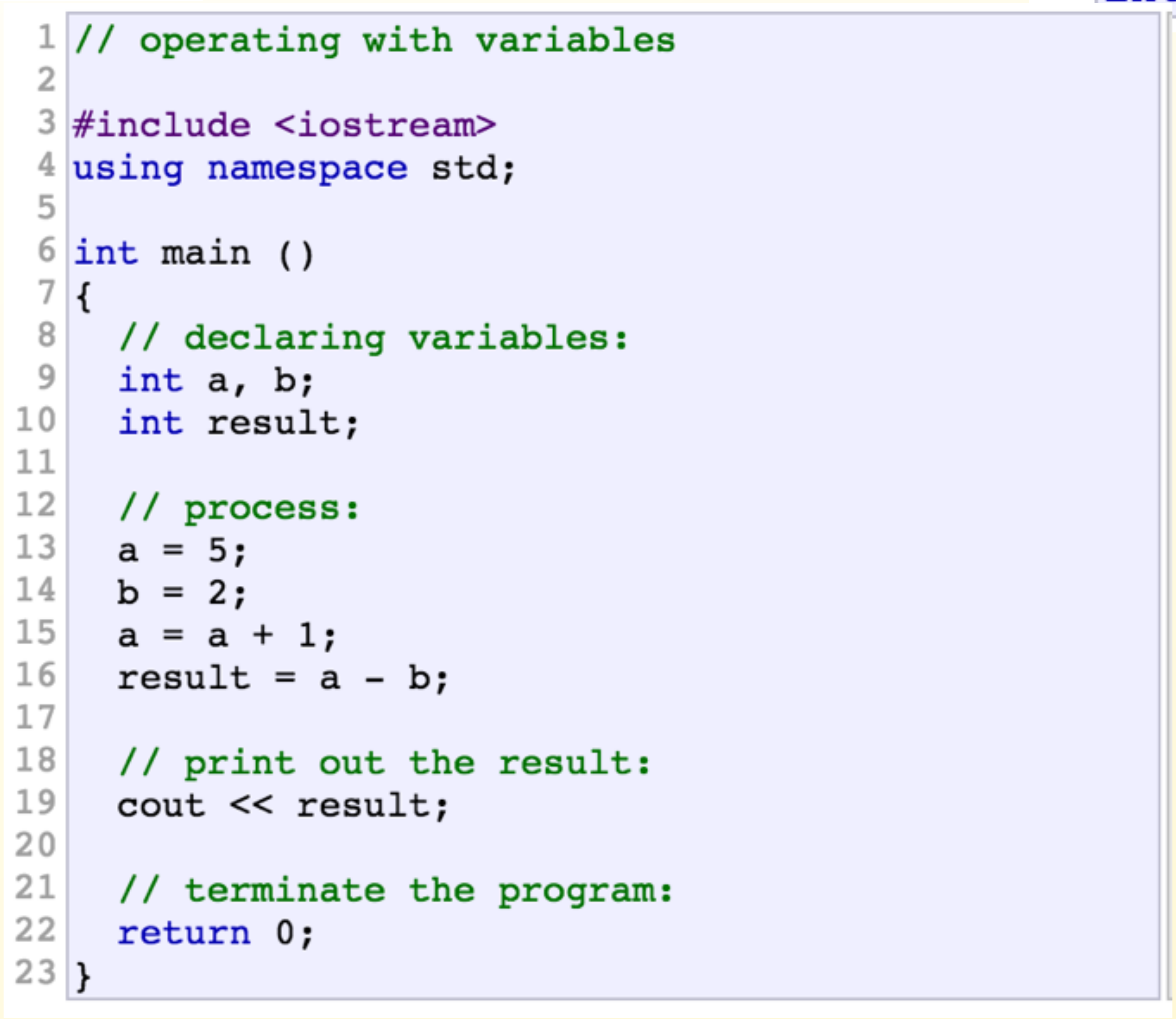
```
int a;
float mynumber;
```



```
int a, b, c;
```

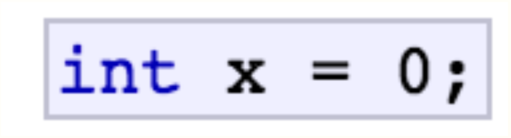


```
int a;
int b;
int c;
```

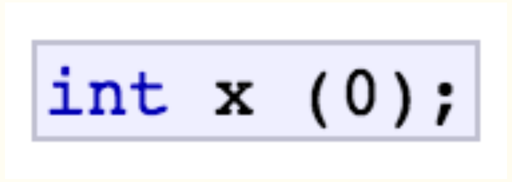


```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

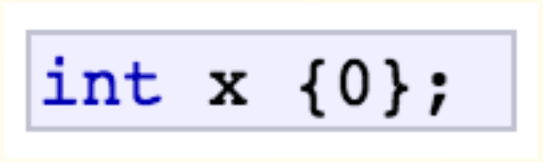
Initialization



```
int x = 0;
```



```
int x (0);
```



```
int x {0};
```

Initialization

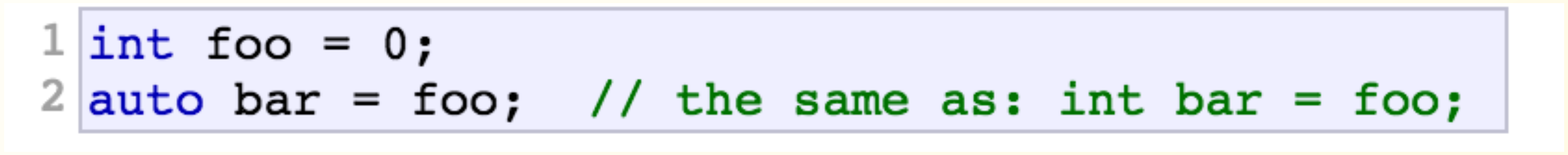
```
int x = 0;
```

```
int x (0);
```

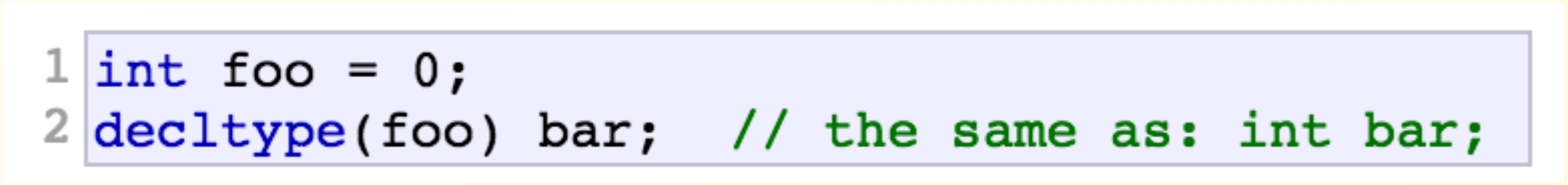
```
int x {0};
```

```
// initialization of variables

#include <iostream>
using namespace std;

int main ()
{
  int a=5;                    // initial value: 5
  int b(3);                   // initial value: 3
  int c{2};                   // initial value: 2
  int result;                 // initial value undetermined

  a = a + b;
  result = a - c;
  cout << result;

  return 0;
}
```

Auto type deduction



```cpp
int foo = 0;
auto bar = foo;   // the same as: int bar = foo;
```



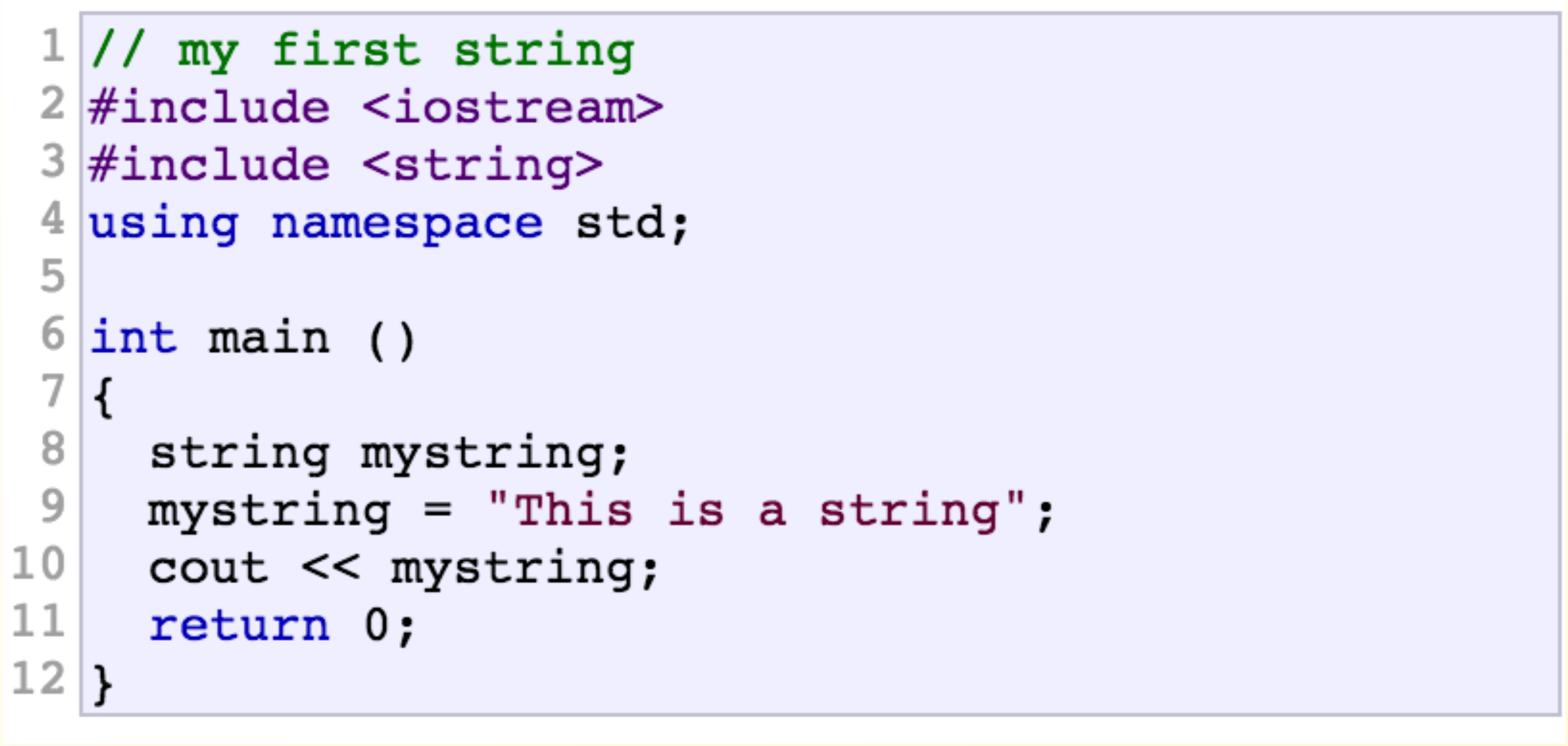
```cpp
int foo = 0;
decltype(foo) bar;   // the same as: int bar;
```

- The keyword auto allows for automatic type deduction and will be really useful when we use templates (see later)

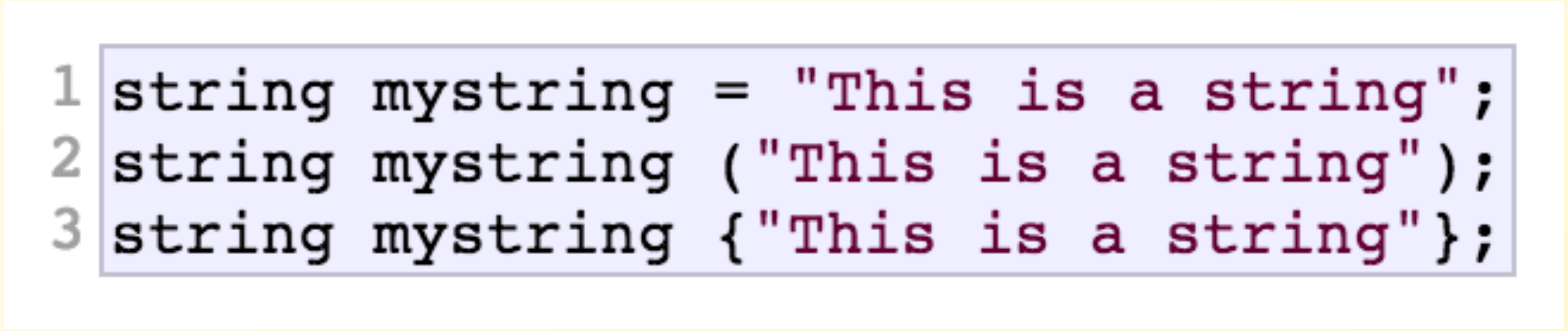
Strings (Intro)

- Previously, arrays of (char *) were used, but they are unsafe. In C++ we use the string type.

Strings (Intro)

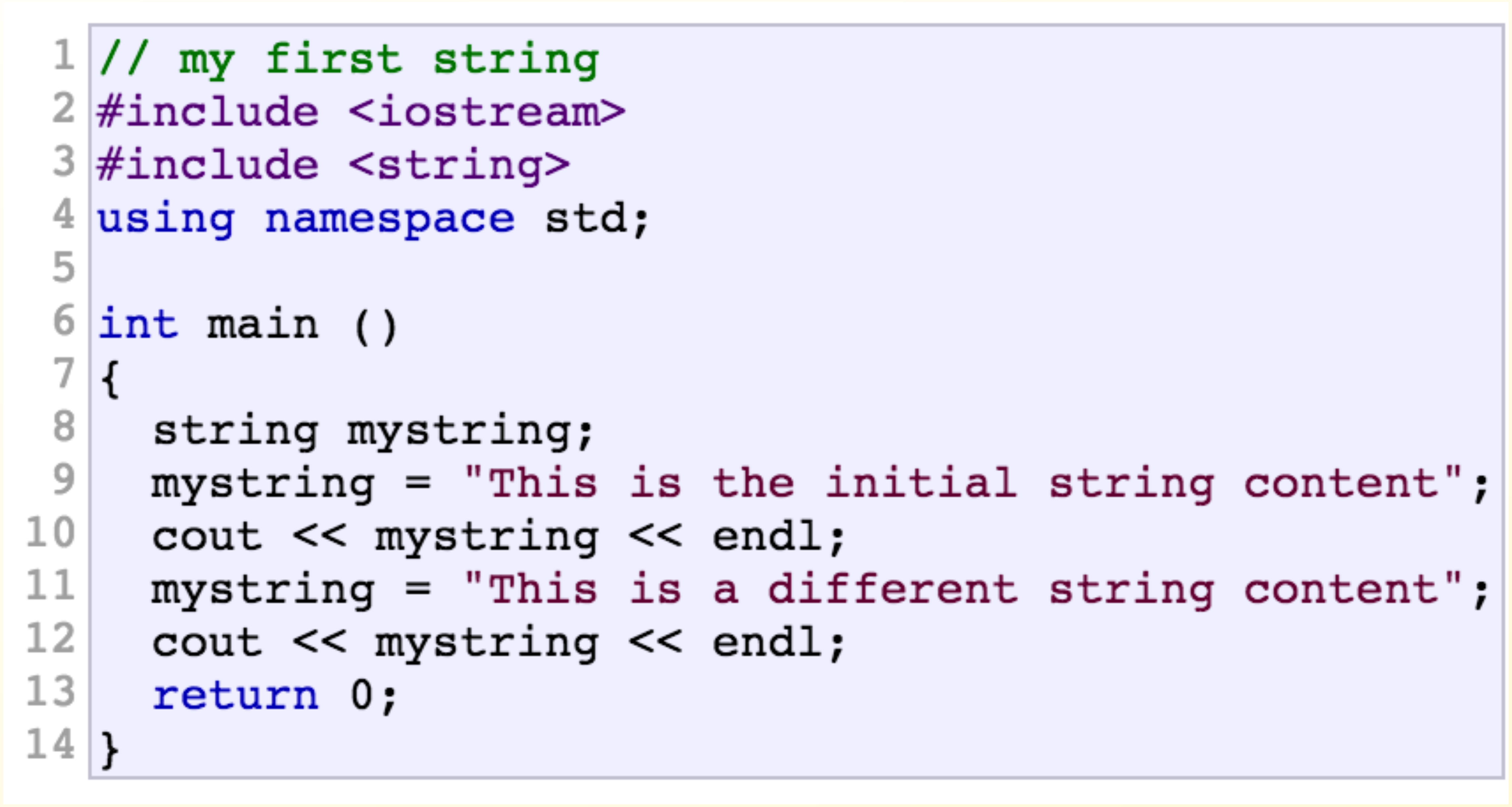


```cpp
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is a string";
  cout << mystring;
  return 0;
}
```



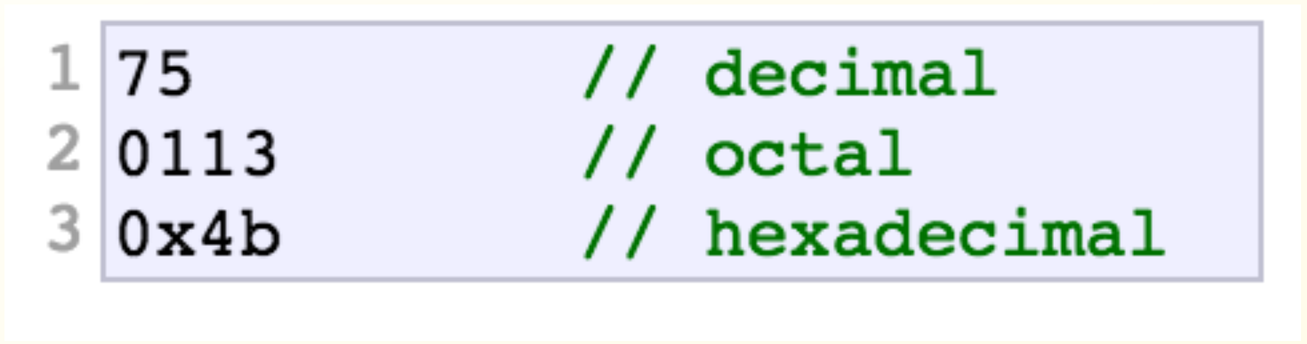
```cpp
string mystring = "This is a string";
string mystring ("This is a string");
string mystring {"This is a string"};
```

Strings (Intro)



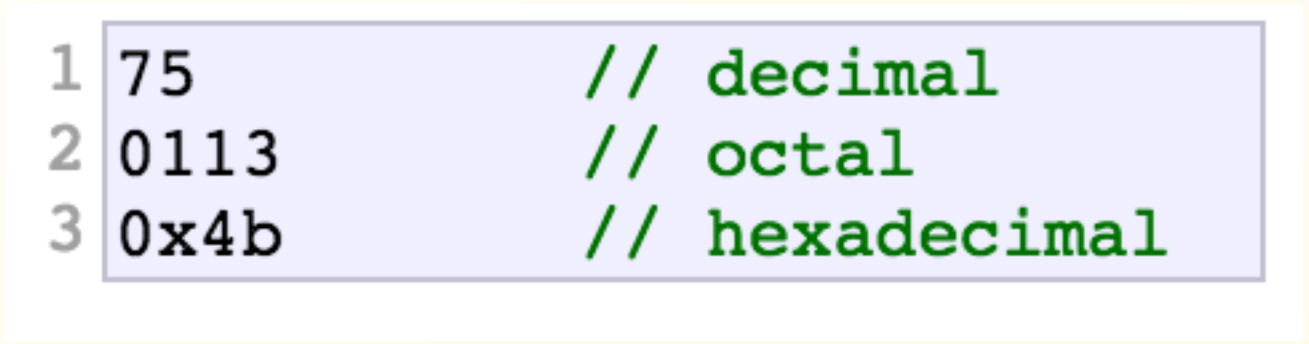
```cpp
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is the initial string content";
  cout << mystring << endl;
  mystring = "This is a different string content";
  cout << mystring << endl;
  return 0;
}
```

Literals

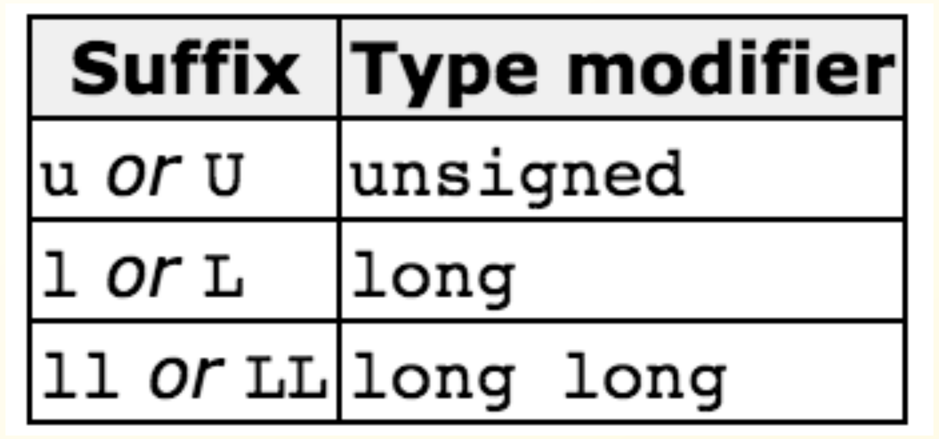


```
75               // decimal
0113             // octal
0x4b             // hexadecimal
```

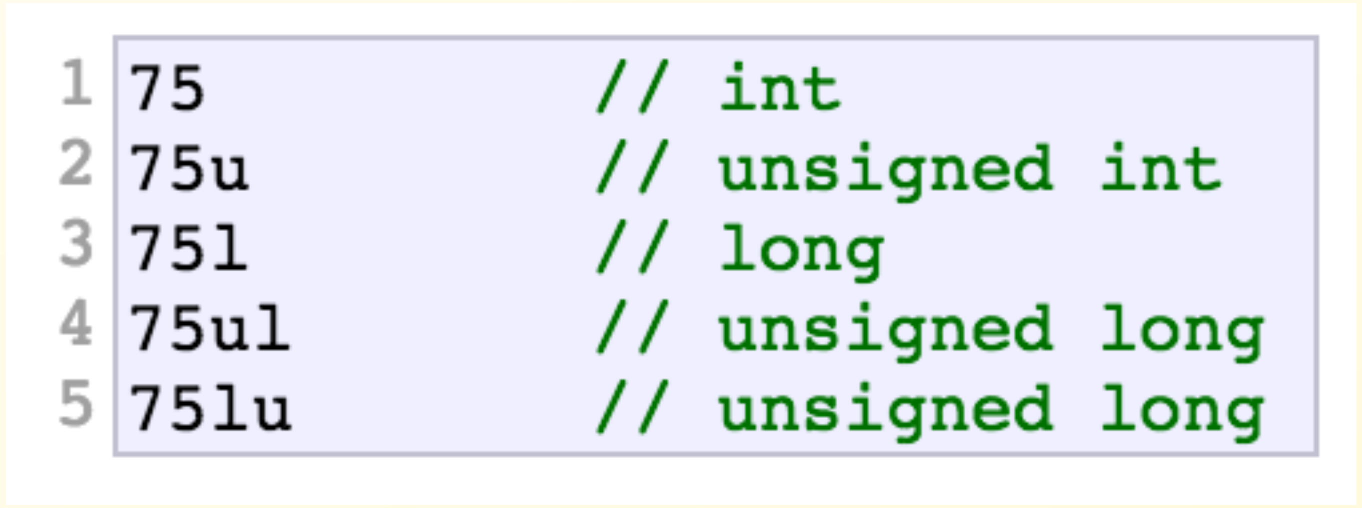
Literals



```
75            // decimal
0113          // octal
0x4b          // hexadecimal
```



Suffix	Type modifier
`u` *or* `U`	`unsigned`
`l` *or* `L`	`long`
`ll` *or* `LL`	`long long`



```
75            // int
75u           // unsigned int
75l           // long
75ul          // unsigned long
75lu          // unsigned long
```

Literals

```
75            // decimal
0113          // octal
0x4b          // hexadecimal
```

Suffix	Type modifier
`u` *or* `U`	`unsigned`
`l` *or* `L`	`long`
`ll` *or* `LL`	`long long`

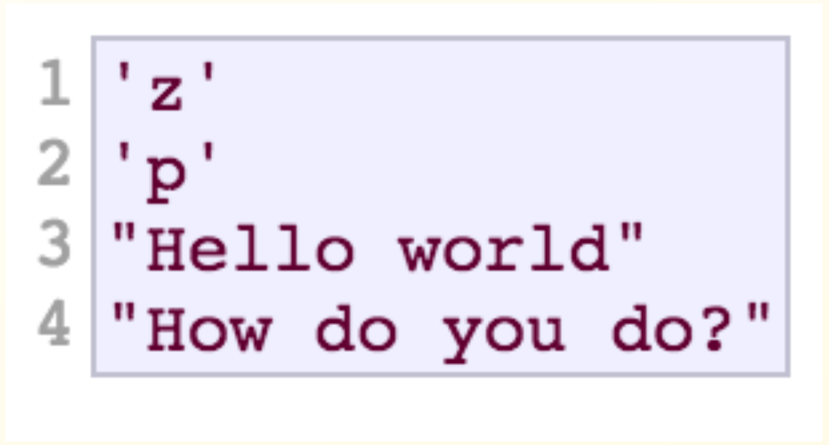
```
75          // int
75u         // unsigned int
75l         // long
75ul        // unsigned long
75lu        // unsigned long
```

Suffix	Type
`f` *or* `F`	`float`
`l` *or* `L`	`long double`

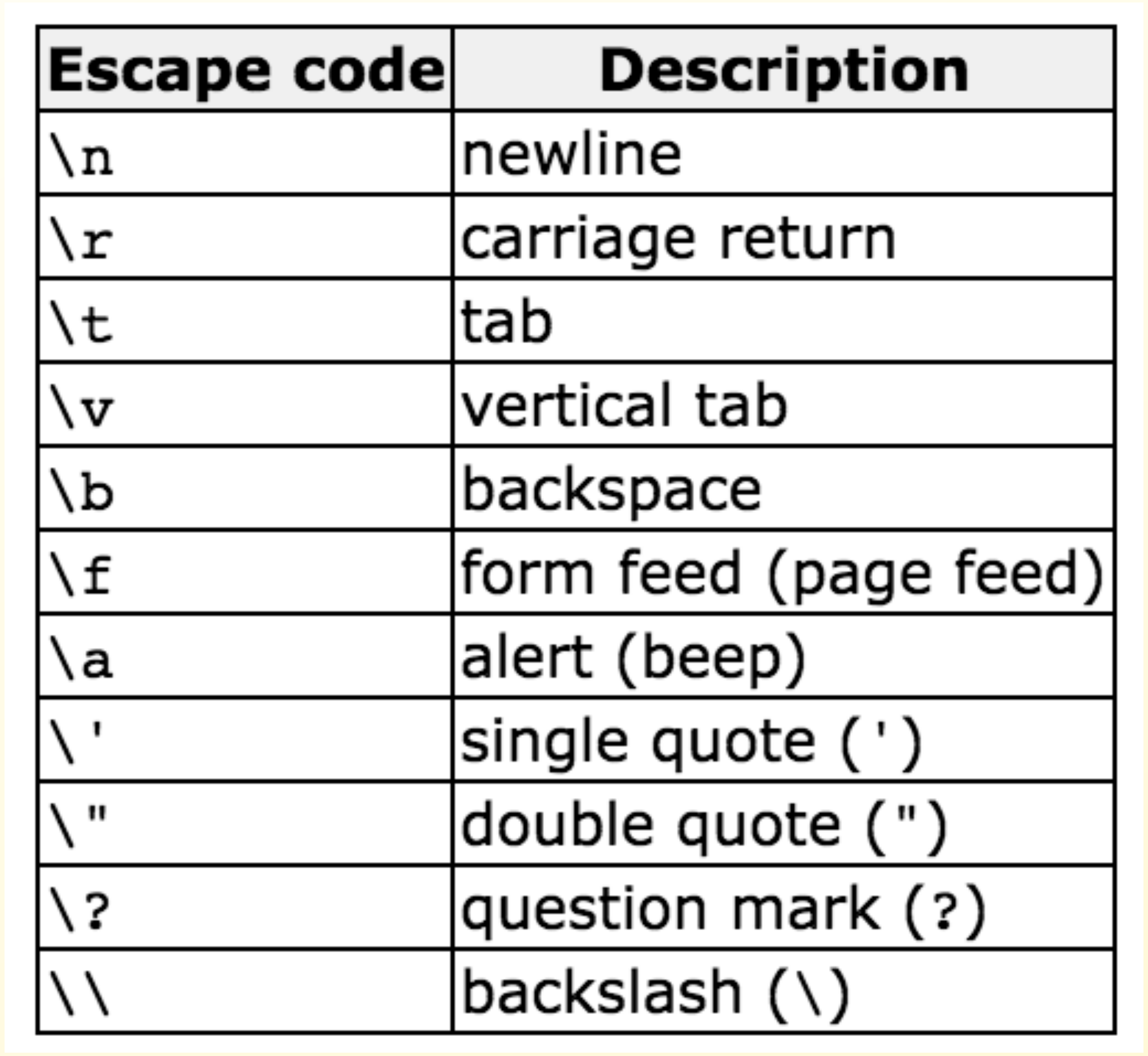
```
3.14159     // 3.14159
6.02e23     // 6.02 x 10^23
1.6e-19     // 1.6 x 10^-19
3.0         // 3.0
```

```
3.14159L    // long double
6.02e23f    // float
```

Literals (cont)

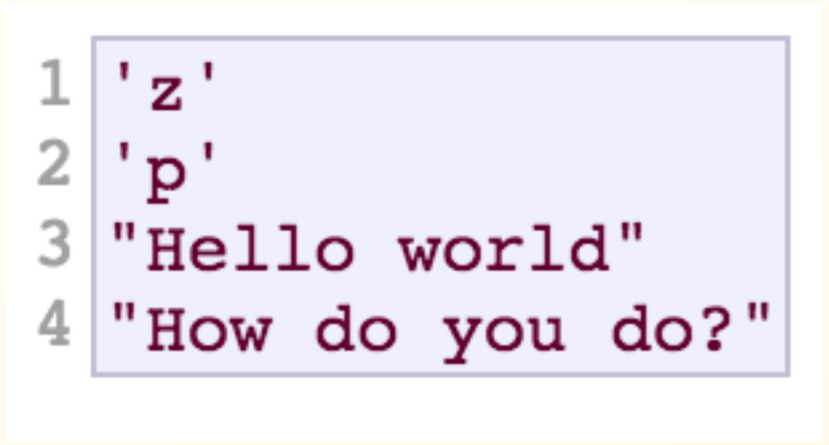


```
'z'
'p'
"Hello world"
"How do you do?"
```

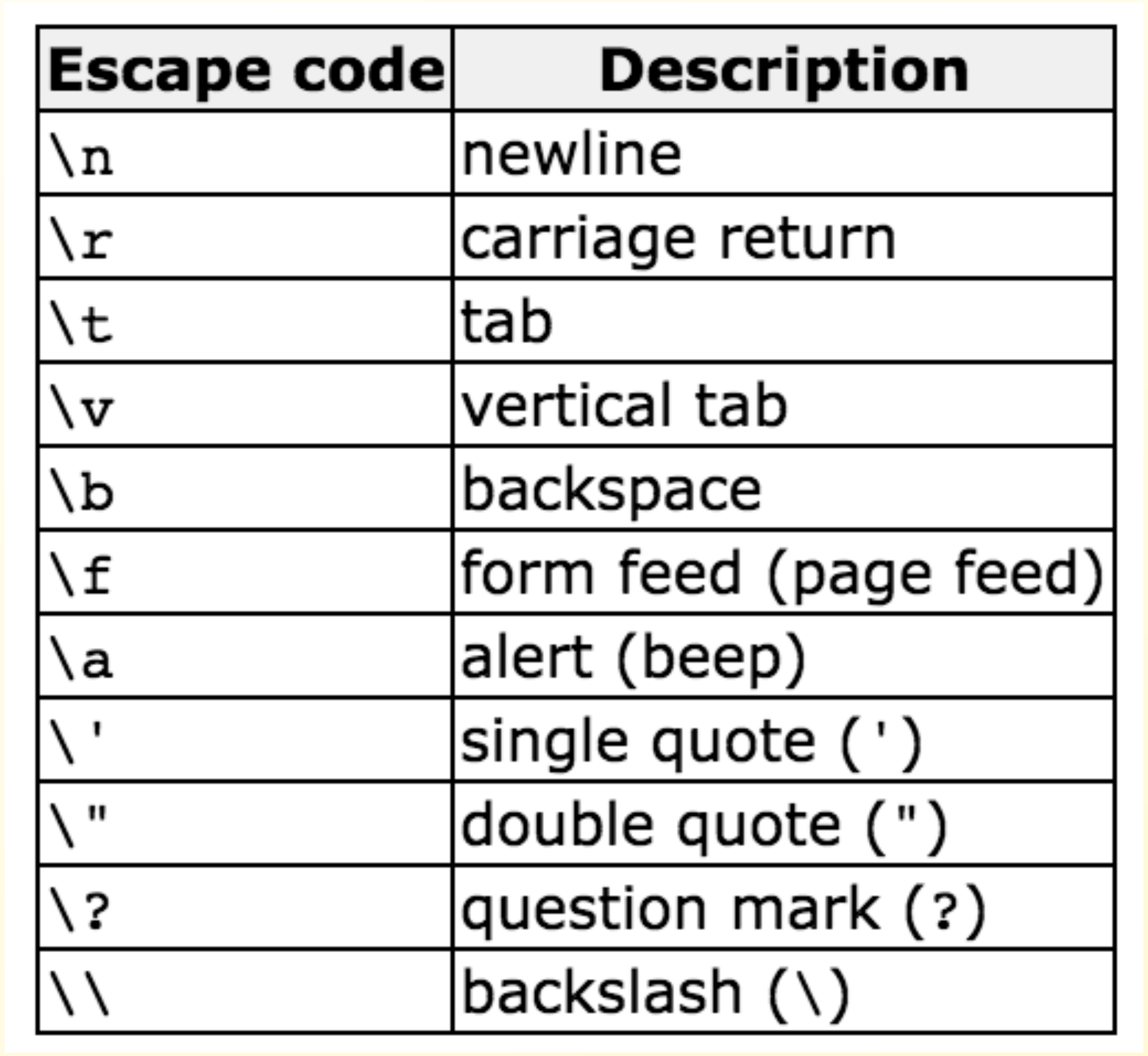


Escape code	Description
`\n`	newline
`\r`	carriage return
`\t`	tab
`\v`	vertical tab
`\b`	backspace
`\f`	form feed (page feed)
`\a`	alert (beep)
`\'`	single quote (`'`)
`\"`	double quote (`"`)
`\?`	question mark (`?`)
`\\`	backslash (`\`)

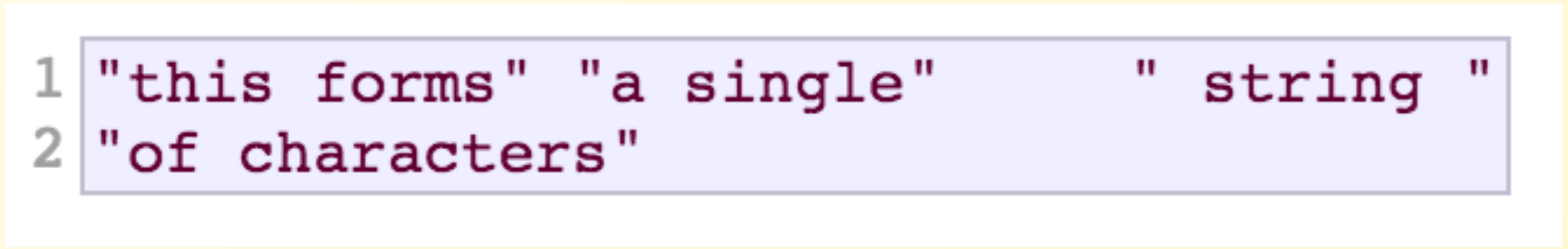
Literals (cont)



```
'z'
'p'
"Hello world"
"How do you do?"
```

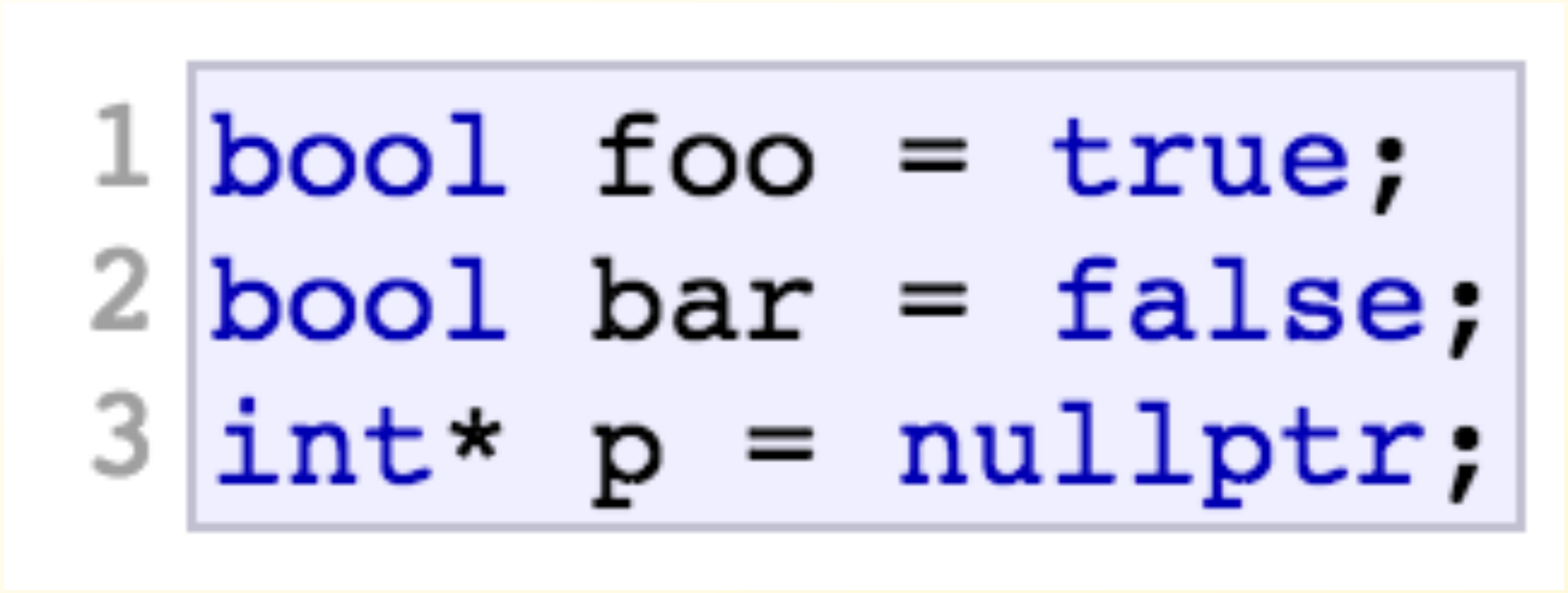


Escape code	Description
`\n`	newline
`\r`	carriage return
`\t`	tab
`\v`	vertical tab
`\b`	backspace
`\f`	form feed (page feed)
`\a`	alert (beep)
`\'`	single quote (`'`)
`\"`	double quote (`"`)
`\?`	question mark (`?`)
`\\`	backslash (`\`)



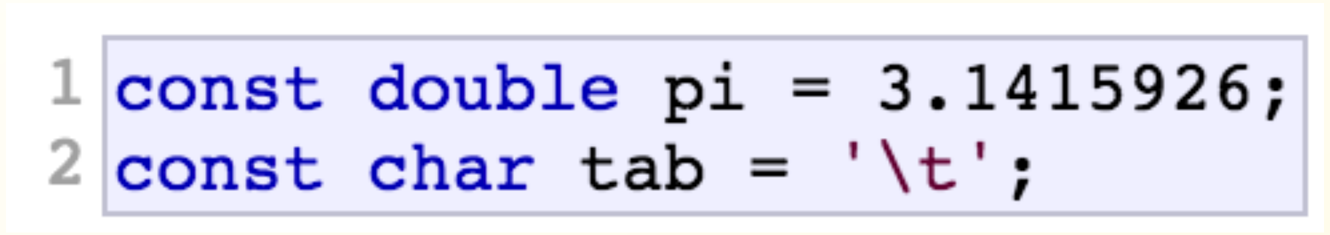
```
"this forms" "a single"      " string "
"of characters"
```

Literals (cont)

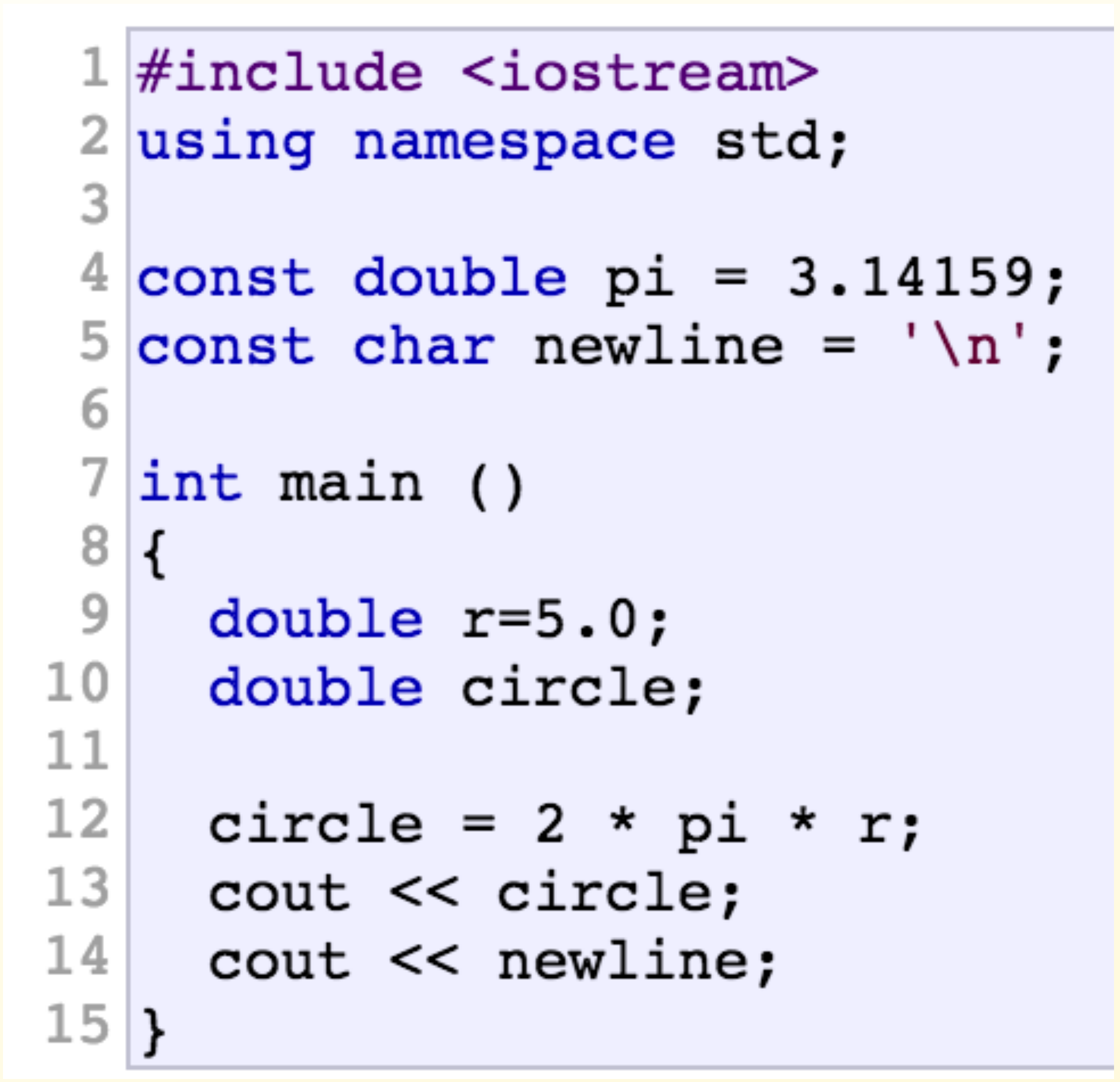


```
bool foo = true;
bool bar = false;
int* p = nullptr;
```

Constant expressions



```
const double pi = 3.1415926;
const char tab = '\t';
```



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

const double pi = 3.14159;
const char newline = '\n';

int main ()
{
  double r=5.0;
  double circle;

  circle = 2 * pi * r;
  cout << circle;
  cout << newline;
}
```

Constant expressions

```cpp
const double pi = 3.1415926;
const char tab = '\t';
```

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

const double pi = 3.14159;
const char newline = '\n';

int main ()
{
  double r=5.0;
  double circle;

  circle = 2 * pi * r;
  cout << circle;
  cout << newline;
}
```

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

#define PI 3.14159
#define NEWLINE '\n'

int main ()
{
  double r=5.0;
  double circle;

  circle = 2 * PI * r;
  cout << circle;
  cout << NEWLINE;

}
```

Review

Review

1. What is meant by the term *prompt*?
2. Which operator do you use to read into a variable?
3. If you want the user to input an integer value into your program for a variable named **number**, what are two lines of code you could write to ask the user to do it and to input the value into your program?
4. What is **\n** called and what purpose does it serve?
5. What terminates input into a string?
6. What terminates input into an integer?
7. How would you write

```
cout << "Hello, ";
cout << first_name;
cout << "!\n";
```

as a single line of code?

8. What is an object?
9. What is a literal?
10. What kinds of literals are there?
11. What is a variable?
12. What are typical sizes for a **char**, an **int**, and a **double**?
13. What measures do we use for the size of small entities in memory, such as **int**s and **string**s?
14. What is the difference between **=** and **==**?

15. What is a definition?

16. What is an initialization and how does it differ from an assignment?

17. What is string concatenation and how do you make it work in C++?

18. Which of the following are legal names in C++? If a name is not legal, why not?

Click here to view code image

```
This_little_pig     This_1_is fine     2_For_1_special
latest thing        the_$12_method      _this_is_ok
MiniMineMine        number             correct?
```

19. Give five examples of legal names that you shouldn't use because they are likely to cause confusion.

20. What are some good rules for choosing names?

21. What is type safety and why is it important?

22. Why can conversion from **double** to **int** be a bad thing?

23. Define a rule to help decide if a conversion from one type to another is safe or unsafe.

Exercises

2 Write a program in C++ that converts from miles to kilometers. Your program should have a reasonable prompt for the user to enter a number of miles. Hint: There are 1.609 kilometers to the mile.

3 Write a program that doesn't do anything, but declares a number of variables with legal and illegal names (such as **int double = 0;**), so that you can see how the compiler reacts.

4 Write a program that prompts the user to enter two integer values. Store these values in **int** variables named **val1** and **val2**. Write your program to determine the smaller, larger, sum, difference, product, and ratio of these values and report them to the user.

5 Modify the program above to ask the user to enter floating-point values and store them in **double** variables. Compare the outputs of the two programs for some inputs of your choice. Are the results the same? Should they be? What's the difference?

6 Write a program that prompts the user to enter three integer values, and then outputs the values in numerical sequence separated by commas. So, if the user enters the values **10 4 6**, the output should be **4, 6, 10**. If two values are the same, they should just be ordered together. So, the input **4 5 4** should give **4, 4, 5**.

7 Do exercise 6, but with three string values. So, if the user enters the values **Steinbeck**, **Hemingway**, **Fitzgerald**, the output should be **Fitzgerald, Hemingway, Steinbeck**.

8 Write a program to test an integer value to determine if it is odd or even. As always, make sure your output is clear and complete. In other words, don't just output **yes** or **no**. Your output should stand alone, like **The value 4 is an even number**. Hint: See the remainder (modulo) operator in §3.4.

9 Write a program that converts spelled-out numbers such as "zero" and "two" into digits, such as 0 and 2. When the user inputs a number, the program should print out the corresponding digit. Do it for the values 0, 1, 2, 3, and 4 and write out **not a number I know** if the user enters something that doesn't correspond, such as **stupid computer!**.

10 Write a program that takes an operation followed by two operands and outputs the result. For example:

```
+ 100 3.14
* 4 5
```

Read the operation into a string called **operation** and use an **if**-statement to figure out which operation the user wants, for example, **if (operation=="+")**. Read the operands into variables of type **double**. Implement this for operations called +, –, *, /, plus, minus, mul, and div with their obvious meanings.

11 Write a program that prompts the user to enter some number of pennies (1-cent coins), nickels (5-cent coins), dimes (10-cent coins), quarters (25-cent coins), half dollars (50-cent coins), and one-dollar coins (100-cent coins). Query the user separately for the number of each size coin, e.g., "How many pennies do you have?" Then your program should print out something like this:

lick here to view code image

```
You have 23 pennies.
You have 17 nickels.
You have 14 dimes.
You have 7 quarters.
You have 3 half dollars.
The value of all of your coins is 573 cents.
```

Make some improvements: if only one of a coin is reported, make the output grammatically correct, e.g., **14 dimes** and **1 dime** (not **1 dimes**). Also, report the sum in dollars and cents, i.e., **$5.73** instead of **573 cents**.

2.1 Write a single C++ statement that prints `"Too many"` if the variable `count` exceeds 100.

2.2 What is wrong with the following code:

a. `cin << count;`

b.
```
if x < y min = x
else min = y;
```

2.3 What is wrong with this code:

```
cout << "Enter n: ";
cin >> n;
if (n < 0)
  cout << "That is negative. Try again." << endl;
  cin >> n;
else
  cout << "o.k. n = " << n << endl;
```

2.4 What is the difference between a reserved word and a standard identifier?

2.5 What is wrong with this code:

```
enum Semester {FALL, SPRING, SUMMER};
enum Season {SPRING, SUMMER, FALL, WINTER};
```

2.6 What is wrong with this code:

```
enum Friends {"Jerry", "Henry", "W.D."};
```

1. What variable type should you use if you want to store a number like 3.1415?
A. `int`
B. `char`
C. `double`
D. `string`

2. Which of the following is the correct operator to compare two variables?
A. `:=`
B. `=`
C. `equal`
D. `==`

3. How do you get access to the string data type?
A. It is built into the language, so you don't need to do anything
B. Since strings are used for IO, you include the `iostream` header file
C. You include the `string` header file
D. C++ doesn’t support strings

4. Which of the following is not a correct variable type?
A. `double`
B. `real`
C. `int`
D. `char`

5. How can you read in an entire line from the user?
A. use `cin>>`
B. Use `readline`
C. use `getline`
D. You cannot do this easily

3. What safeguards does C++ provide to keep you from exceeding the limits of an integer type?

4. What is the distinction between 33L and 33?

5. Consider the two C++ statements that follow:

   ```
   char grade = 65;
   char grade = 'A';
   ```

   Are they equivalent?

6. How could you use C++ to find out which character the code 88 represents? Come up with at least two ways.

7. Assigning a `long` value to a `float` can result in a rounding error. What about assigning `long` to `double`? `long long` to `double`?

8. Evaluate the following expressions as C++ would:

   a. 8 * 9 + 2

   b. 6 * 3 / 4

   c. 3 / 4 * 6

   d. 6.0 * 3 / 4

   e. 15 % 4

9. Suppose `x1` and `x2` are two type `double` variables that you want to add as integers and assign to an integer variable. Construct a C++ statement for doing so. What if you want to add them as type `double` and then convert to `int`?

10. What is the variable type for each of the following declarations?

    a. `auto cars = 15;`

    b. `auto iou = 150.37f;`

    c. `auto level = 'B';`

    d. `auto crat = U'/U00002155';`

    e. `auto fract = 8.25f/2.5;`