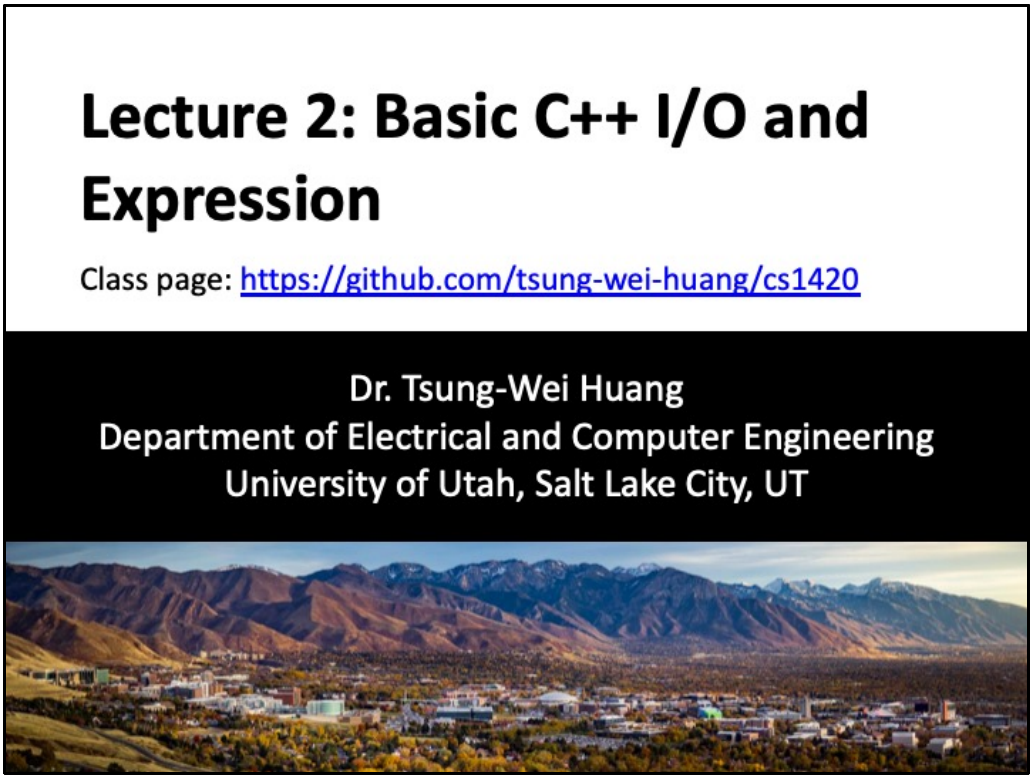


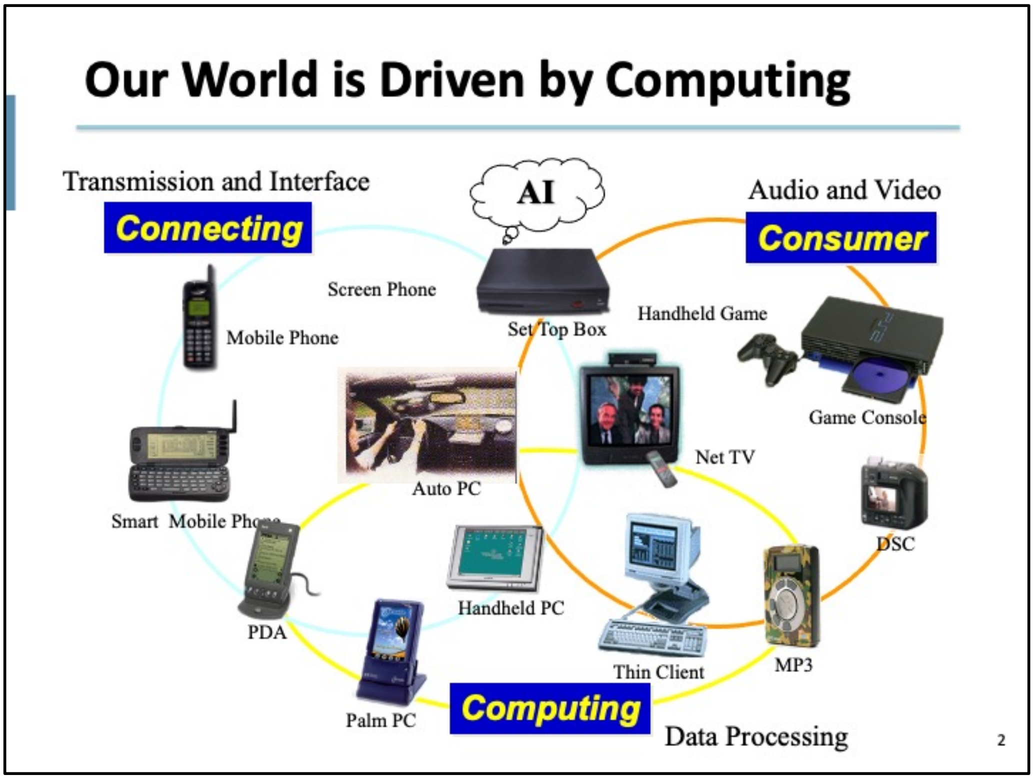
Lecture 2: Basic C++ I/O and Expression

Class page: https://github.com/tsung-wei-huang/cs1420

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

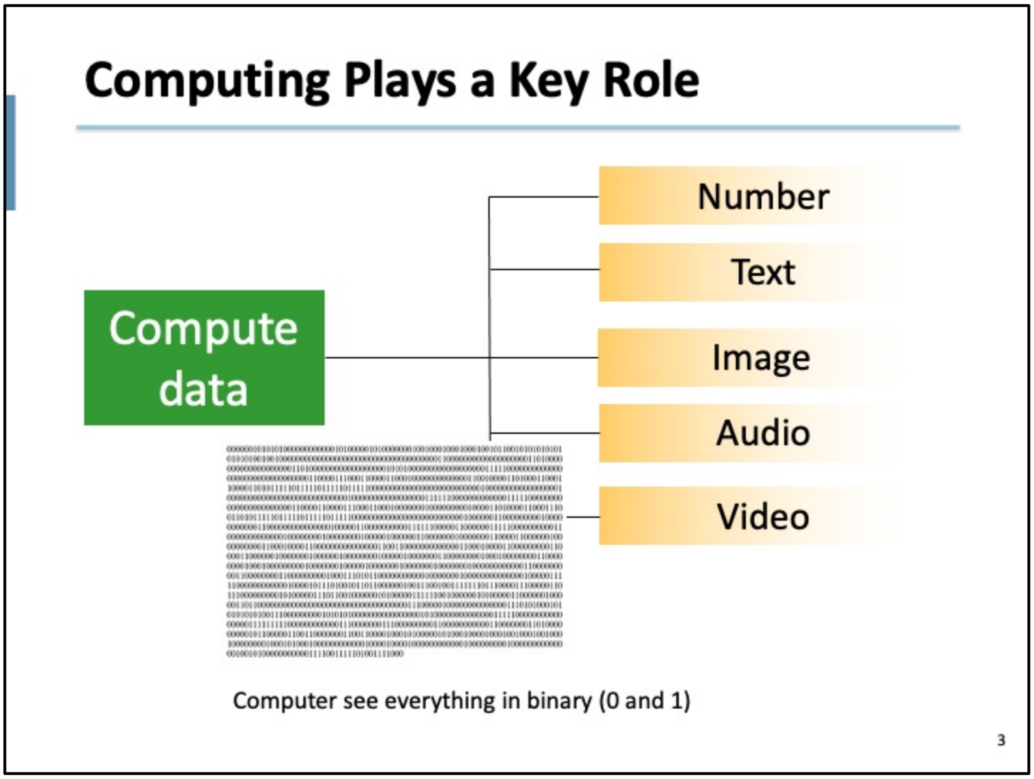
Morning everyone

Today we are going to look at some basic C++ input and output and expression.



Our World is Driven by Computing

Last time, we talked about we are living in a highly connected world. The internet has become the knowledge hub for almost everyone in the world to get connected to each other and the newest trends. We use the internet to do almost all kinds of jobs. And a critical component to support this success is the computing. We want to compute data. we want to extract some useful information out of the data.



Computing Plays a Key Role

Compute data

Number

Text

Image

Audio

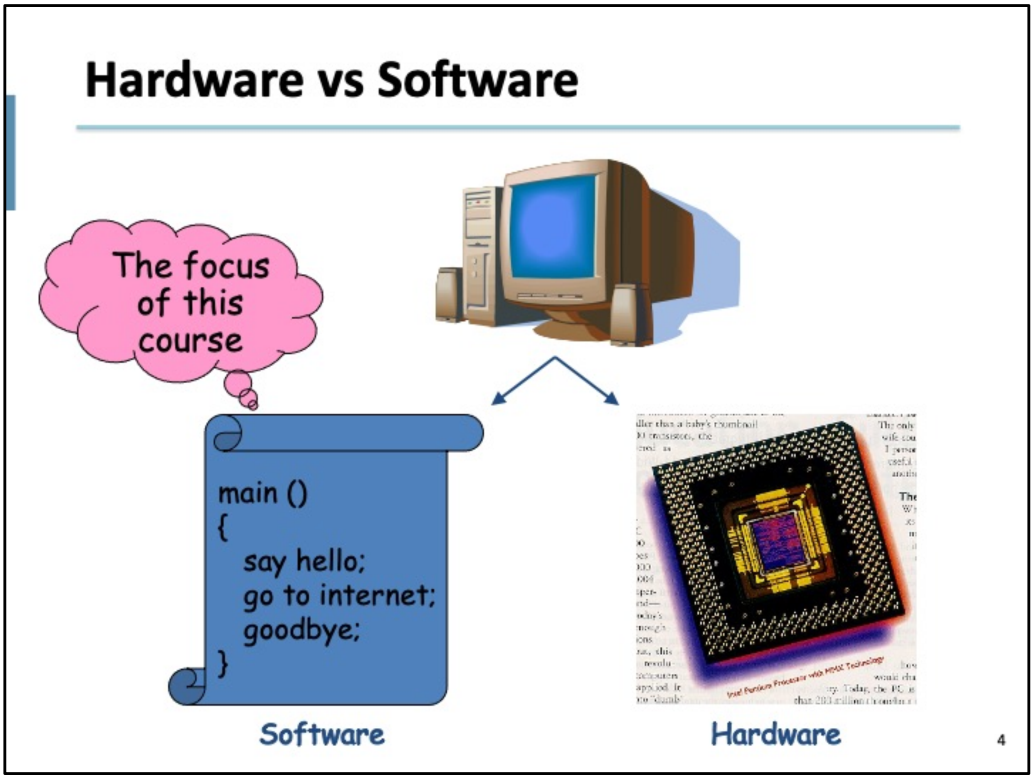
Video

Computer see everything in binary (0 and 1)

3

Data can be numbers (like you phone number), or plain text, images (which are the most commonly used data), audio and video (like youtube). Whichever data you have, we need to compute the data to extract some useful information. We cannot do this by hand, because today's data is so huge and so enormous that we cannot rely on human to do all these things for us. We need to rely on computers because they can compute data much faster than human.

This is indeed very challenging. In the world of computers, everything is digitized. everything is just zero or one. That is what computers see everything in his world. Of course, we must avoid this non-human readable information because no one will be able to interpret zero and one.



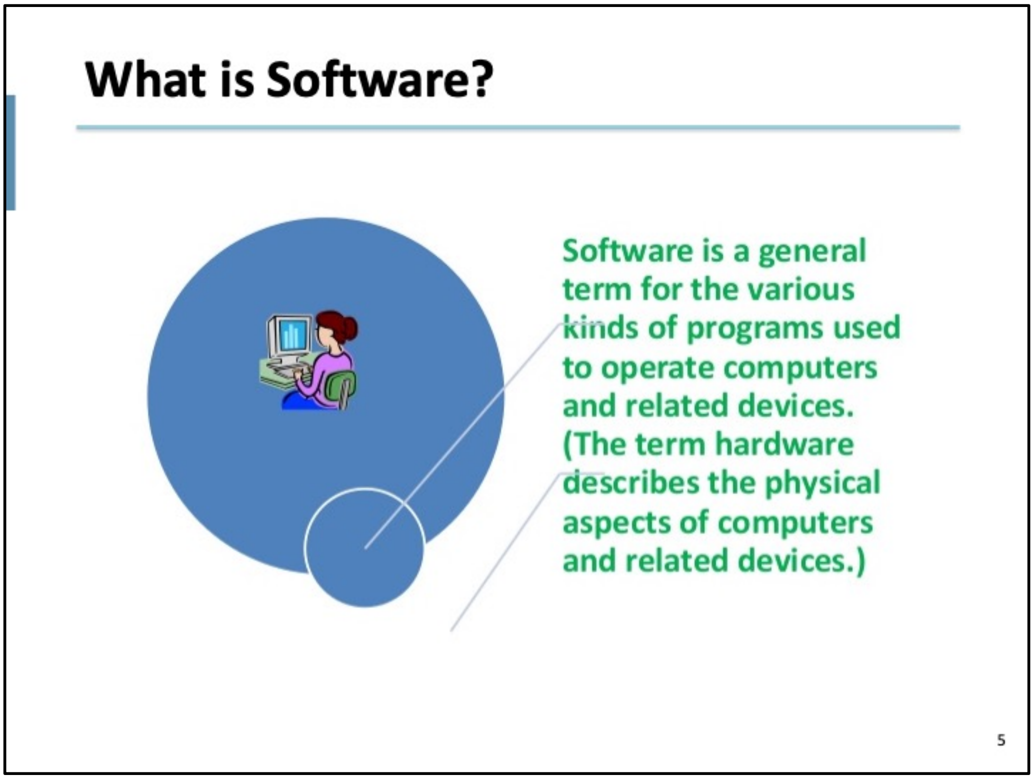
Hardware vs Software

```
main ()
{
   say hello;
   go to internet;
   goodbye;
}
```

During the computation, there are two phases, one software and another one hardware. Software is the major layer for people to talk to computer. HArdware is the layer for computer to actually work on your program and run it on computing resources.

By the term you can probably guess, software is more flexible because when you get something wrong, you can just rewrite it, wipe it out and restart everything again.

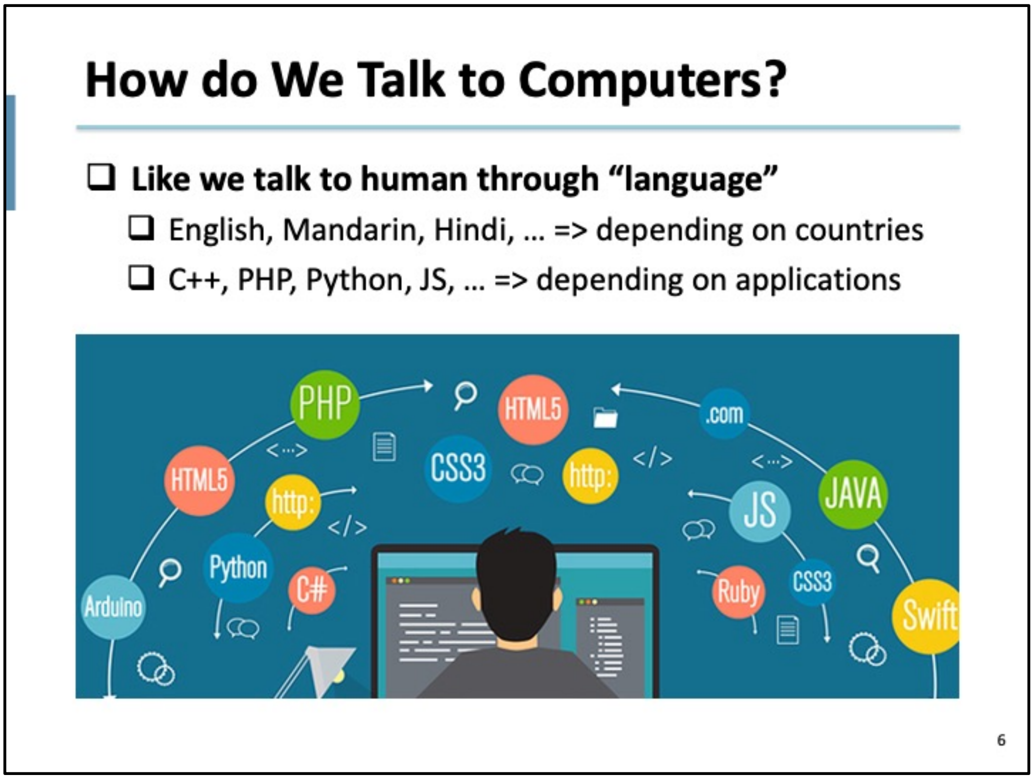
However, it's a different story on the hardware. When you have a malfunction in hardware, it's million dollars. And the manufacturing process in today's hardware devices are so complicated that you won't afford to have too many errors. This is the fundamental difference between software and hardware.



What is Software?

Software is a general term for the various kinds of programs used to operate computers and related devices. (The term hardware describes the physical aspects of computers and related devices.)

What is software? ...



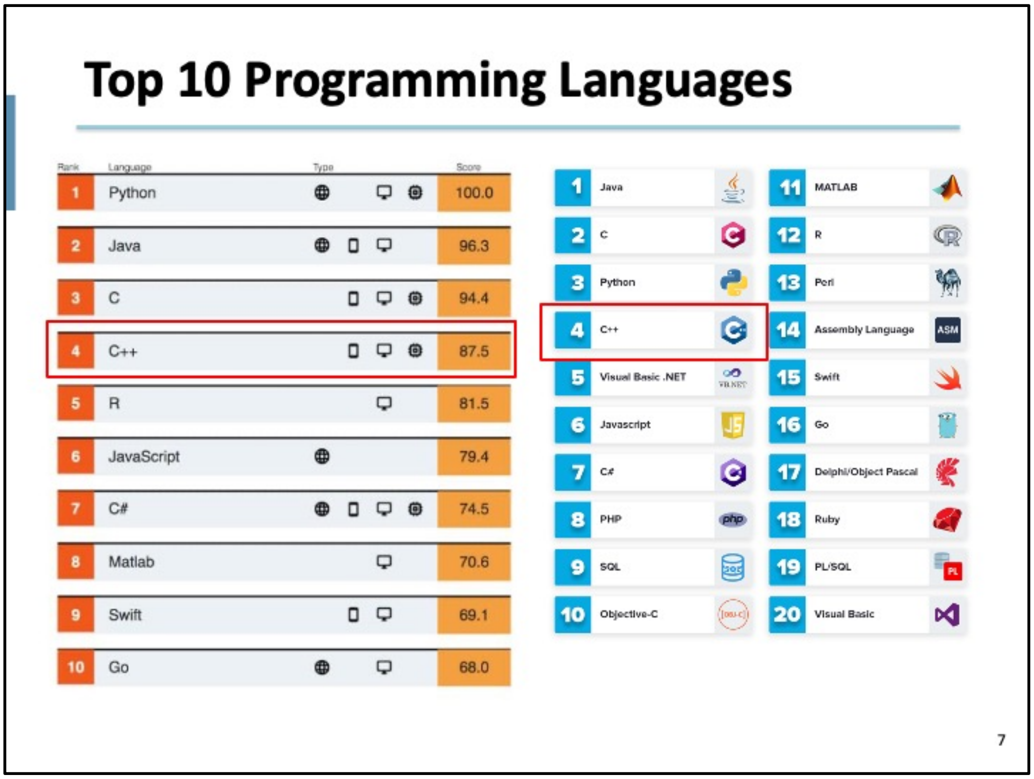
How do We Talk to Computers?

- ❑ **Like we talk to human through "language"**
  - ❑ English, Mandarin, Hindi, … => depending on countries
  - ❑ C++, PHP, Python, JS, … => depending on applications

In order to operate computers, we must first talk to them. But how do we talk to computers?

Like we talk to human, we use language. We talk to people from different countries using different languages, English, Mandarin, Hindi, and so on, it all depends on countries. For example, if the person comes from the US or the UK, we talk in English. If the person comes from China, we talk in Mandarin.

In computer programming, this "country aspect" translates to "applications". Depending on applications, we have different programming languages, such as C++, Php, python, and javascript. For instance, if you are doing data science, you need quick turnaround time and Python is the best choice. If you are doing computer engineering or electrical engineering, C or C++ is the right language. So, it really depends on the application you target. And the application is driven by the marekt.



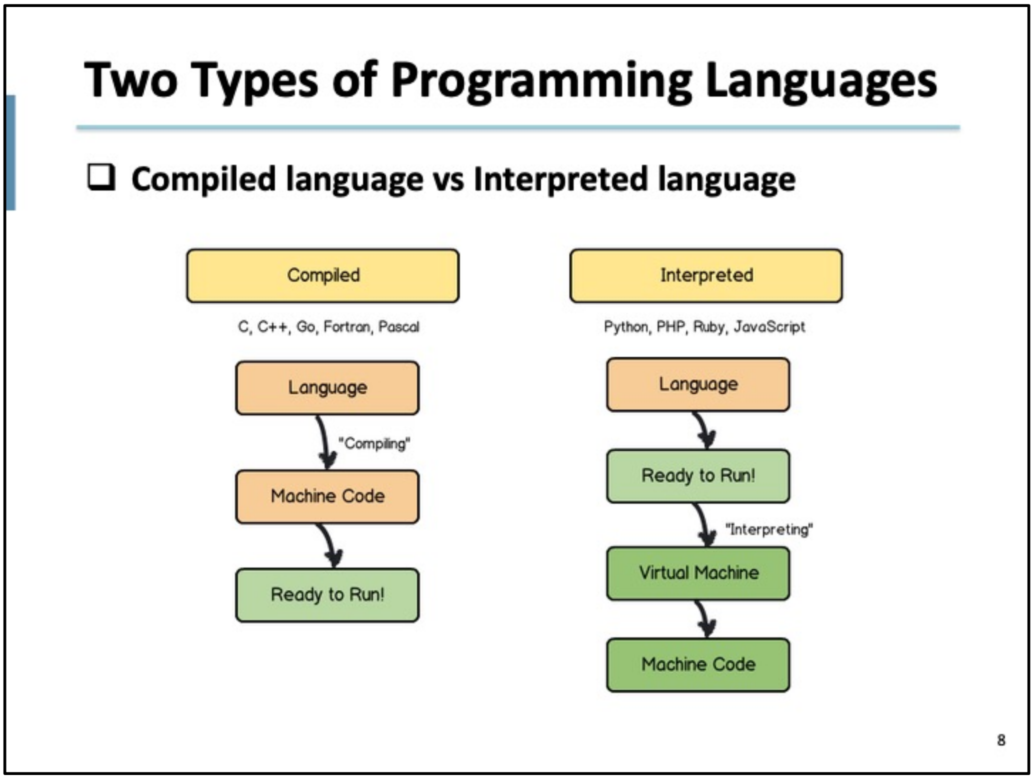
Top 10 Programming Languages

Rank	Language	Type				Score
1	Python	🌐		🖥	⚙	100.0
2	Java	🌐	📱	🖥		96.3
3	C		📱	🖥	⚙	94.4
4	C++		📱	🖥	⚙	87.5
5	R			🖥		81.5
6	JavaScript	🌐				79.4
7	C#	🌐	📱	🖥	⚙	74.5
8	Matlab			🖥		70.6
9	Swift		📱	🖥		69.1
10	Go	🌐		🖥		68.0

1	Java	11	MATLAB
2	C	12	R
3	Python	13	Perl
4	C++	14	Assembly Language
5	Visual Basic .NET	15	Swift
6	Javascript	16	Go
7	C#	17	Delphi/Object Pascal
8	PHP	18	Ruby
9	SQL	19	PL/SQL
10	Objective-C	20	Visual Basic

7

This slides shows the top 10 programming languages from multiple resources in 2019. Different tech reports have different results and some of the reports favor their preferred languages. But you can clearly see some of the languages appear in all reports, such as Java, Python, C, and C++, because they are just too important to be ignored. Languages like Java, Python, and C++ dominate the entire software industry.



Two Types of Programming Languages

❑ **Compiled language vs Interpreted language**

Compiled

C, C++, Go, Fortran, Pascal

Language

"Compiling"

Machine Code

Ready to Run!

Interpreted

Python, PHP, Ruby, JavaScript

Language

Ready to Run!

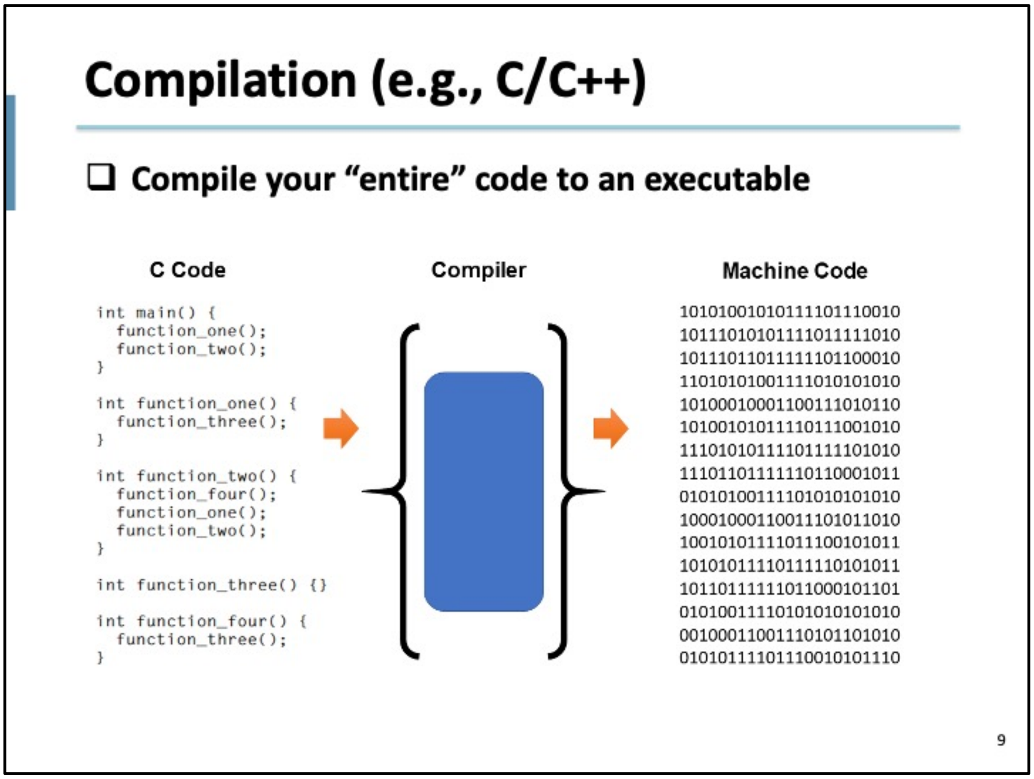
"Interpreting"

Virtual Machine

Machine Code

Roughly speaking, there are two types of programming languages: compiled language and interpreted language. The key difference is:

- the program written in compiled language cannot run directly (...
- the program written in interpreted language can run directly (...



Compilation (e.g., C/C++)

- ❑ Compile your "entire" code to an executable

C Code

```
int main() {
  function_one();
  function_two();
}

int function_one() {
  function_three();
}

int function_two() {
  function_four();
  function_one();
  function_two();
}

int function_three() {}

int function_four() {
  function_three();
}
```

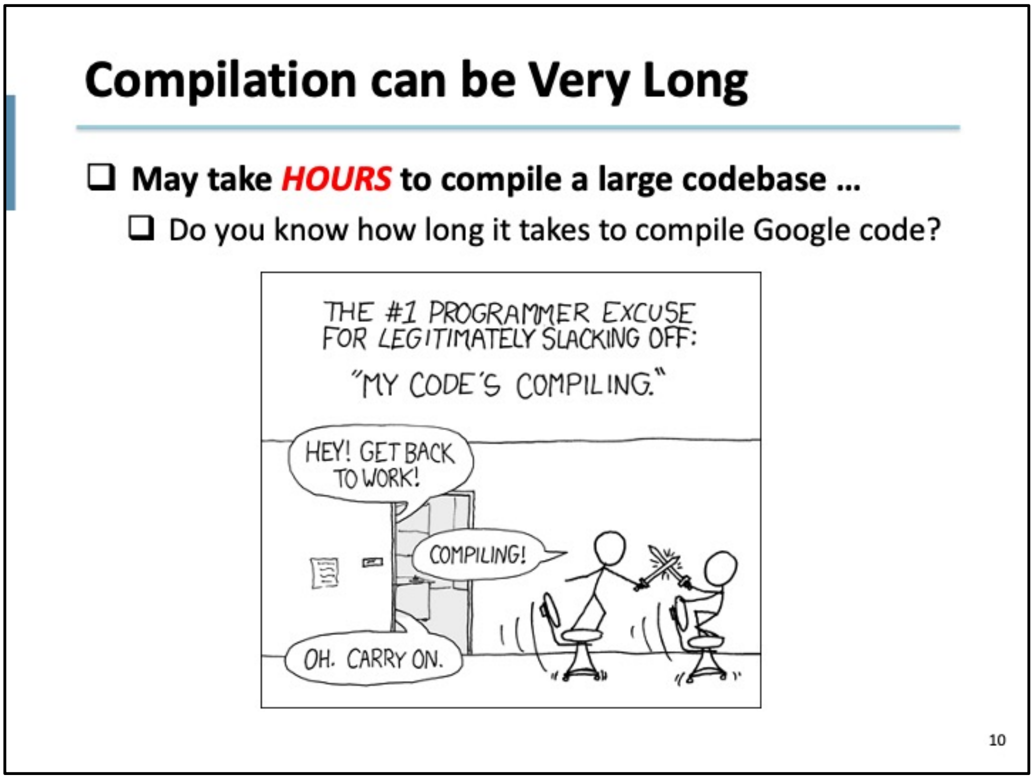
Machine Code

```
10101001010101111101110010
10111010101111011111010
10111011011111101100010
11010101001111010101010
10100010001100111010110
10100101011110111001010
11101010111101111101010
11101101111110110001011
01010100111101010101010
10001000110011101011010
10010101111011100101011
10101011110111110101011
10110111111011000101101
01010011110101010101010
00100011001110101101010
01010111101110010101110
```

9

Let's look at an example of compilation using C programming language. On the left-hand side we have a source code written in C (you don't need to understand its syntax but just recognize it is a C program). In order to run the code, you need to have a compiler first, and invoke the compiler to compile the source code into machine-readable code. Machine-readable code is also called "assembly". Basically, they are just a bunch of zero and ones. These machine code are highly optimized based on your platform. You know, different platforms have different assembly architectures, such as Windows book, MACbook, Google chrome book, and the compiler can generate the machine code optimized for that platform.

So, a key take-away here is, compiled language can give you the best performance because the program you generated is highly optimized on a particular platform.



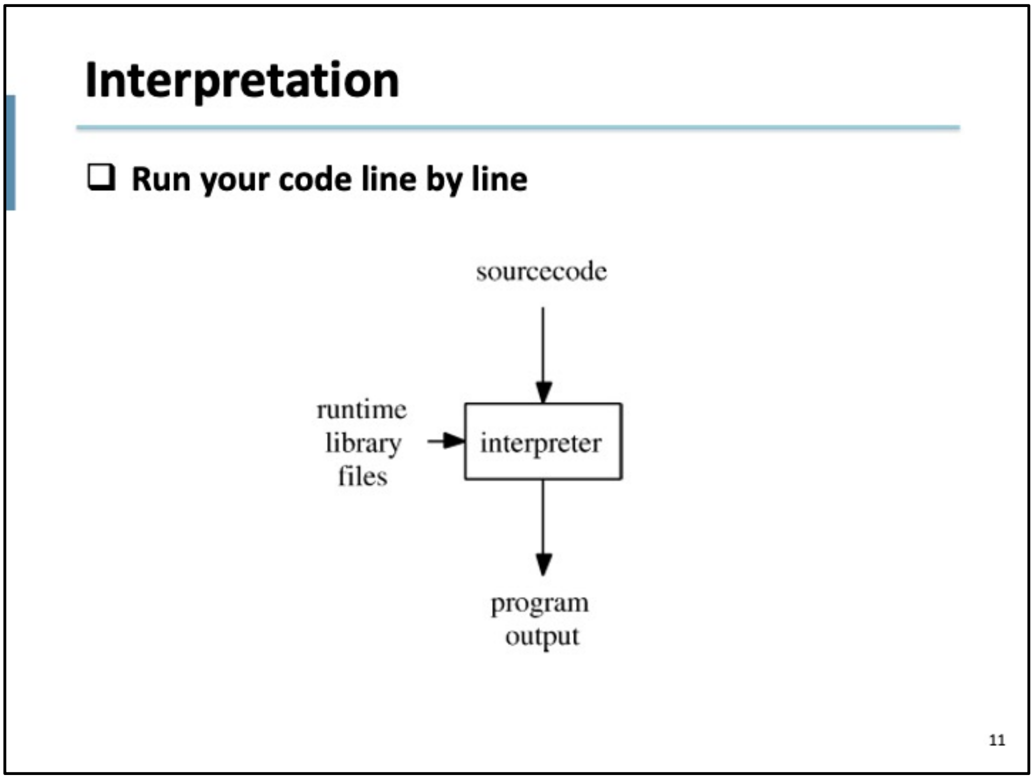
Compilation can be Very Long

- ❑ **May take *HOURS* to compile a large codebase …**
  - ❑ Do you know how long it takes to compile Google code?

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

The major downside of compiled language is compilation can be very long. It may take hours to compile a large codebase ...

In a company I work before, if we want to find someone for coffee, we don't say "let's go for coffee" but "hey, get your code compiled". It's funny because it is really long, and everytime I change something I need to get it compiled first, before I can run it to test my results. This is the biggest problem of compiled language.



Interpretation

❑ **Run your code line by line**

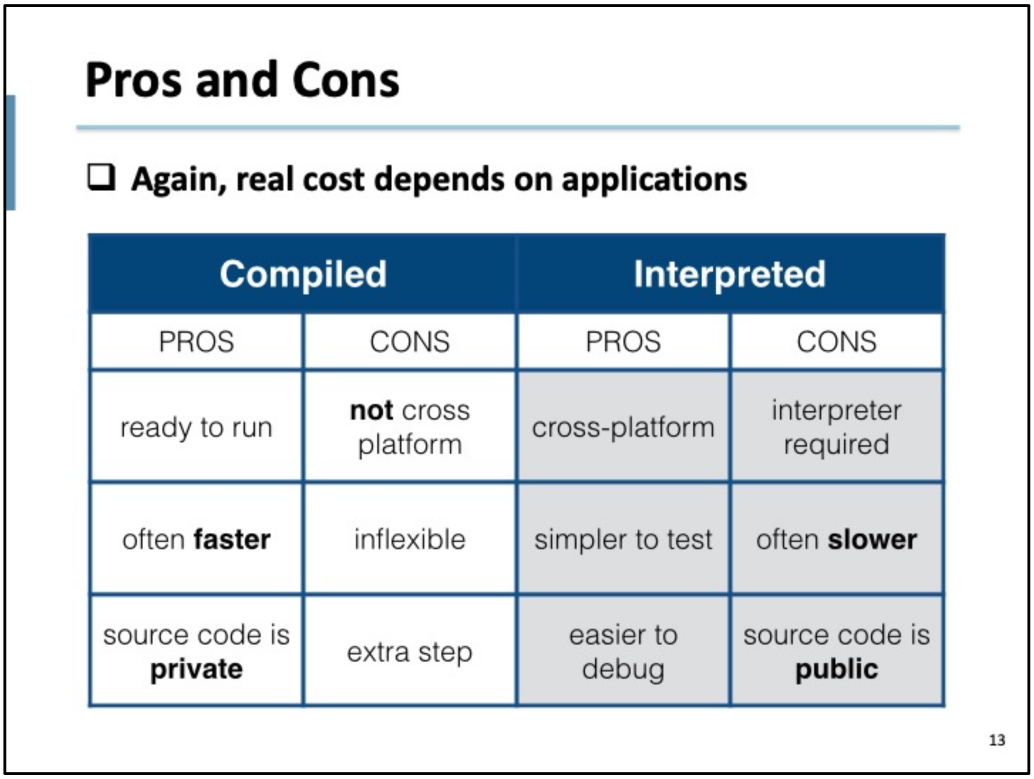
Different from compilation, interpretation runs your code line by line. It doesn't need to see the entire code at once like compiled language but can run it line by line. This means, if there are syntax errors, like you write something wrong, the interpreter will not encounter that errors until it executes that line of code. So you have a much faster turnaround time to test your code during the development phase.

Interpretation can be Very Slow

Slowest things on earth:

Python

You probably can guess, interpretation can be very slow - because it runs everything line-by-line without compilation that optimizes the control flow and code. Many people are joking with Python which is a very famous interpreted language, the speed is like turtle.



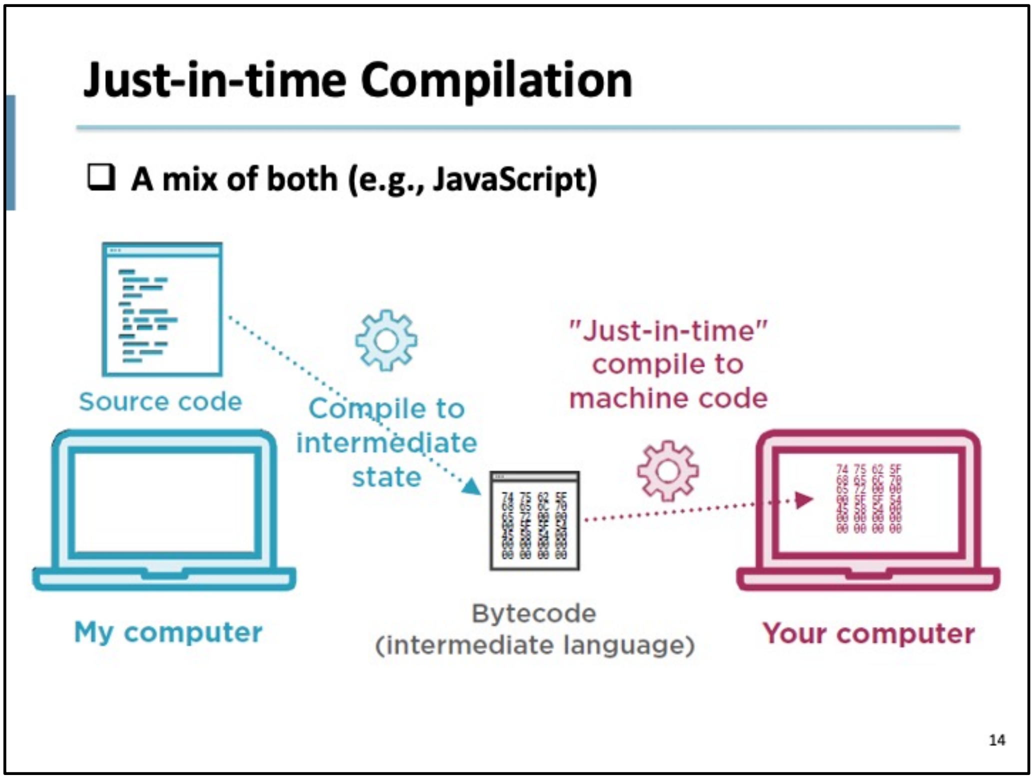
Pros and Cons

- ❑ **Again, real cost depends on applications**

Compiled		**Interpreted**	
PROS	CONS	PROS	CONS
ready to run	**not** cross platform	cross-platform	interpreter required
often **faster**	inflexible	simpler to test	often **slower**
source code is **private**	extra step	easier to debug	source code is **public**

13

This table summarizes the pros and cons between compiled language and interpreted language.



Just-in-time Compilation

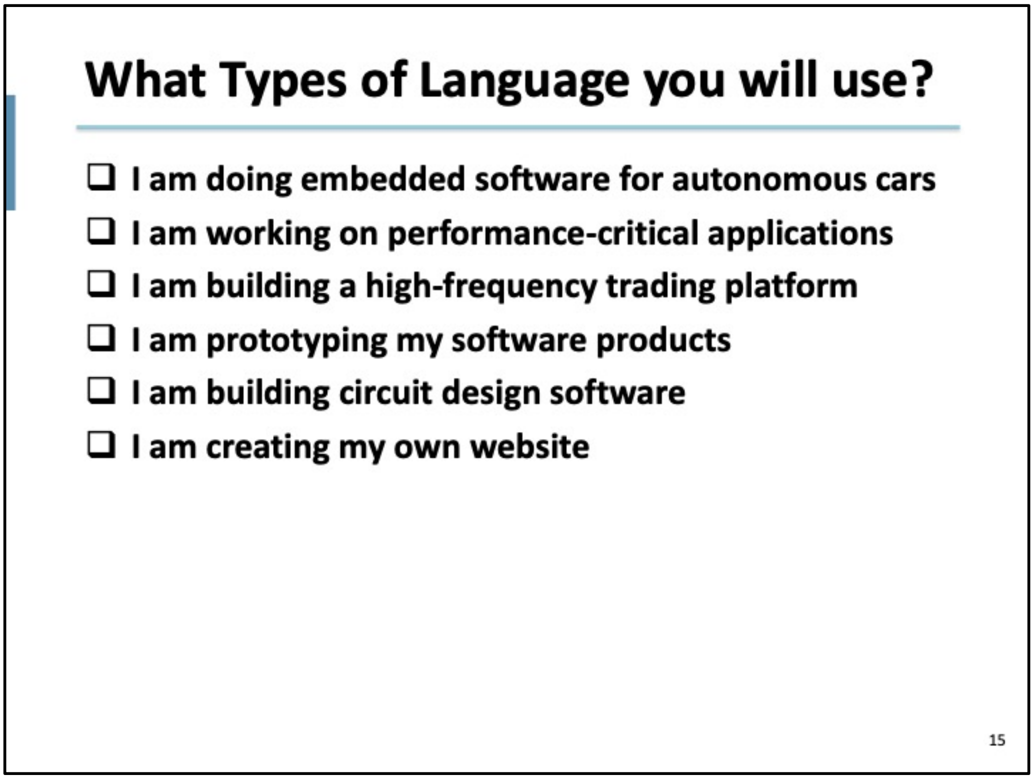
❑ A mix of both (e.g., JavaScript)

Source code
Compile to intermediate state
"Just-in-time" compile to machine code
My computer
Bytecode (intermediate language)
Your computer
74 75 62 5F

14

In fact, there is a recent technology becoming more and more popular called Just-in-time compilation or JIT for short. Just in time compilation essentially combines the advantages of compiled language and interpreted language. What it does is, it starts with interpretation, and keeps running the code. While running the code, it does some analysis on the code and may invoke a native compiler to generate assembly to optimize some performance critical blocks. So you can see this type of technology is very useful for applications that have regular or iterative process. (explore...

Of course, Just in time compilation also inherits the downsides of both, so it is used in certian applications and is not replacing interpreter or compiler.

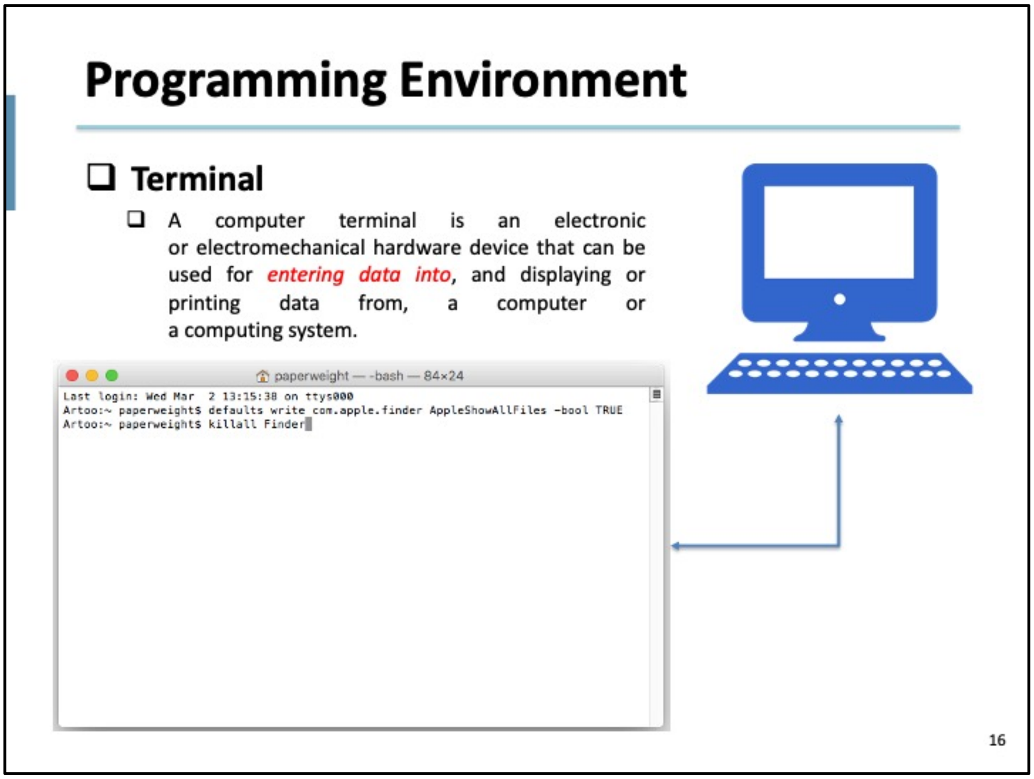


What Types of Language you will use?

- □ I am doing embedded software for autonomous cars
- □ I am working on performance-critical applications
- □ I am building a high-frequency trading platform
- □ I am prototyping my software products
- □ I am building circuit design software
- □ I am creating my own website

15

Let's stop here for 3 minutes and please take a look at the following scenarios and think about which kind of language you may need to use.



Programming Environment

❑ Terminal

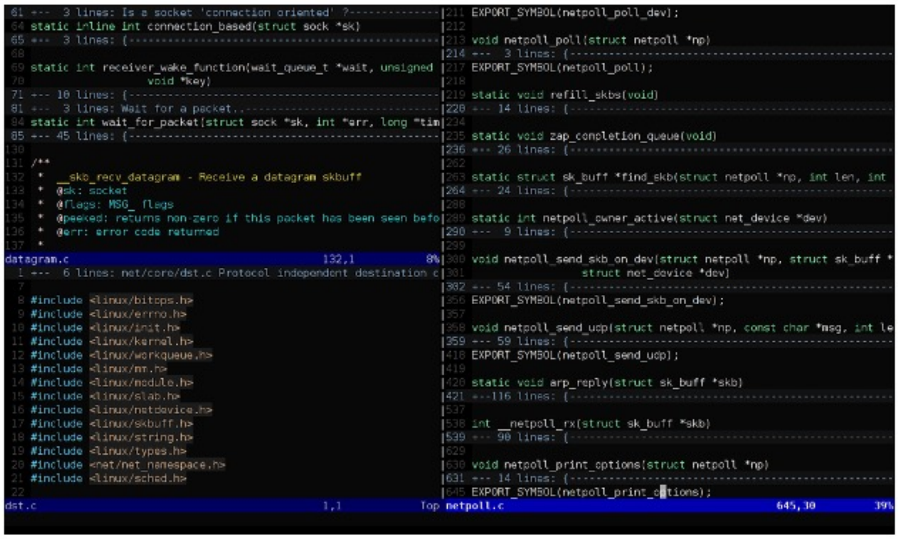
- ❑ A computer terminal is an electronic or electromechanical hardware device that can be used for *entering data into*, and displaying or printing data from, a computer or a computing system.

```
Last login: Wed Mar  2 13:15:38 on ttys000
Artoo:~ paperweight$ defaults write com.apple.finder AppleShowAllFiles -bool TRUE
Artoo:~ paperweight$ killall Finder
```

Now we know the two types of programming language, what about the programming environment. In this course, we will use "terminal" to write your program. A computer terminal is an ....

Essentially, a terminal is the "most basic" interface for you to operate computer without graphic interface. It is very simple and we will use terminal to wirte programs most of time.

Editor (I normally use VIM)



```
+---  3 lines: Is a socket 'connection oriented' ?--------------
static inline int connection_based(struct sock *sk)
+---  3 lines: {------------------------------------------------

static int receiver_wake_function(wait_queue_t *wait, unsigned
                   void *key)
+--- 10 lines: {------------------------------------------------
+---  3 lines: Wait for a packet..------------------------------
static int wait_for_packet(struct sock *sk, int *err, long *tim
+--- 45 lines: {------------------------------------------------

/**
 *   __skb_recv_datagram - Receive a datagram skbuff
 *   @sk: socket
 *   @flags: MSG_ flags
 *   @peeked: returns non-zero if this packet has been seen befo
 *   @err: error code returned
 *
datagram.c                                          132,1      8%
```

```
+---  6 lines: net/core/dst.c Protocol independent destination c

#include <linux/bitops.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/workqueue.h>
#include <linux/mm.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/netdevice.h>
#include <linux/skbuff.h>
#include <linux/string.h>
#include <linux/types.h>
#include <net/net_namespace.h>
#include <linux/sched.h>

dst.c                                               1,1        Top
```

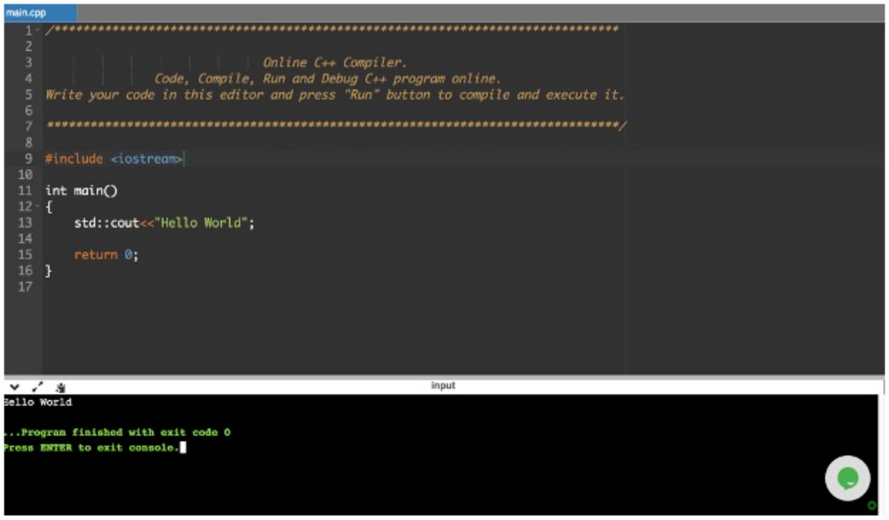
```
EXPORT_SYMBOL(netpoll_poll_dev);

void netpoll_poll(struct netpoll *np)
+---  3 lines: {-------------------------------------------------
EXPORT_SYMBOL(netpoll_poll);

static void refill_skbs(void)
+--- 14 lines: {-------------------------------------------------

static void zap_completion_queue(void)
+--- 26 lines: {-------------------------------------------------

static struct sk_buff *find_skb(struct netpoll *np, int len, int
+--- 24 lines: {-------------------------------------------------

static int netpoll_owner_active(struct net_device *dev)
+---  9 lines: {-------------------------------------------------

void netpoll_send_skb_on_dev(struct netpoll *np, struct sk_buff *
                 struct net_device *dev)
+--- 54 lines: {-------------------------------------------------
EXPORT_SYMBOL(netpoll_send_skb_on_dev);

void netpoll_send_udp(struct netpoll *np, const char *msg, int le
+--- 59 lines: {-------------------------------------------------
EXPORT_SYMBOL(netpoll_send_udp);

static void arp_reply(struct sk_buff *skb)
+---116 lines: {-------------------------------------------------

int __netpoll_rx(struct sk_buff *skb)
+--- 90 lines: {-------------------------------------------------

void netpoll_print_options(struct netpoll *np)
+--- 14 lines: {-------------------------------------------------
EXPORT_SYMBOL(netpoll_print_options);
netpoll.c                                        645,30       39%
```

17

Once you open a terminal that connects to a computer, you will need an editor to write programs. An editor is like microsoft word or powerpoint. It lets you type in code and define syntax highligh to colorize different keywords so it makes everything easier to recognize. In this course, we will use "vim" - for many of you who have never used it before, the learning curve may be a bit high, but once you overcome the hurdle, it becomes very easy and general. For now, I am still using vim and I believe most engineers doing C++ programming are using vim today.

There are also online C++ Editors

❑ https://www.onlinegdb.com/online_c++_compiler

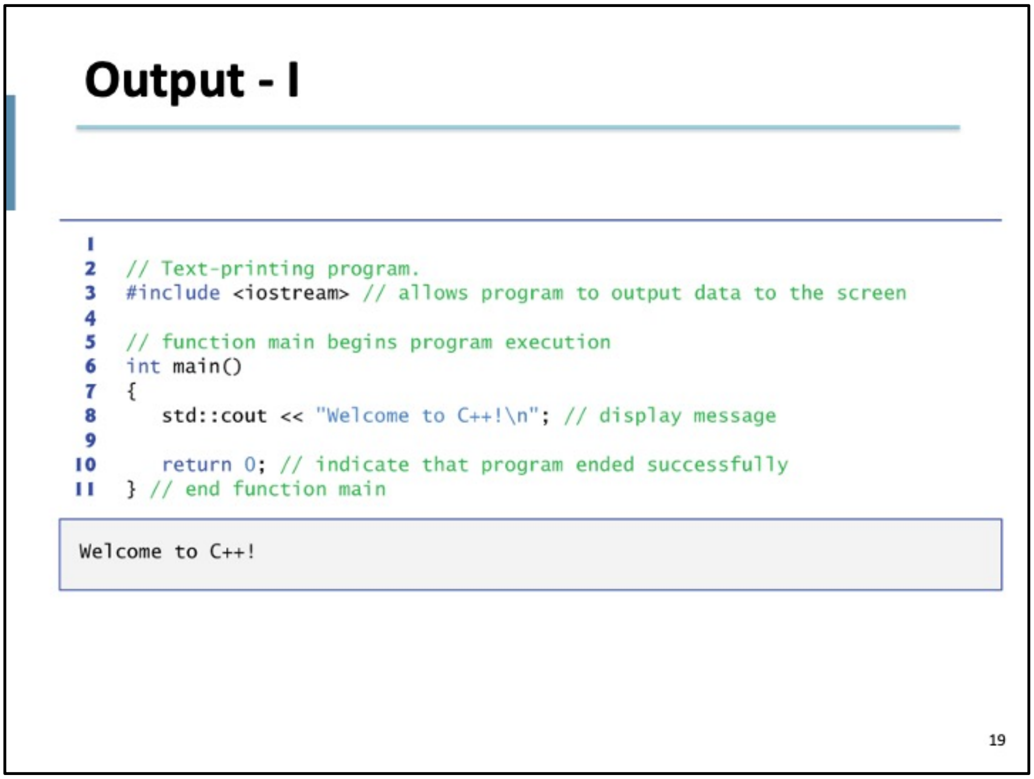


```
/******************************************************************************

                              Online C++ Compiler.
               Code, Compile, Run and Debug C++ program online.
Write your code in this editor and press "Run" button to compile and execute it.

*******************************************************************************/

#include <iostream>

int main()
{
    std::cout<<"Hello World";

    return 0;
}
```

```
Hello World

...Program finished with exit code 0
Press ENTER to exit console.
```

18

There are also online editor that you can use to write C++ code. This is useful when you don't have access to a terminal or a machine but you want to write C++ code and test it in a quick cycle.



Output - I

```
// 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
```

```
Welcome to C++!
```

Any question so far?

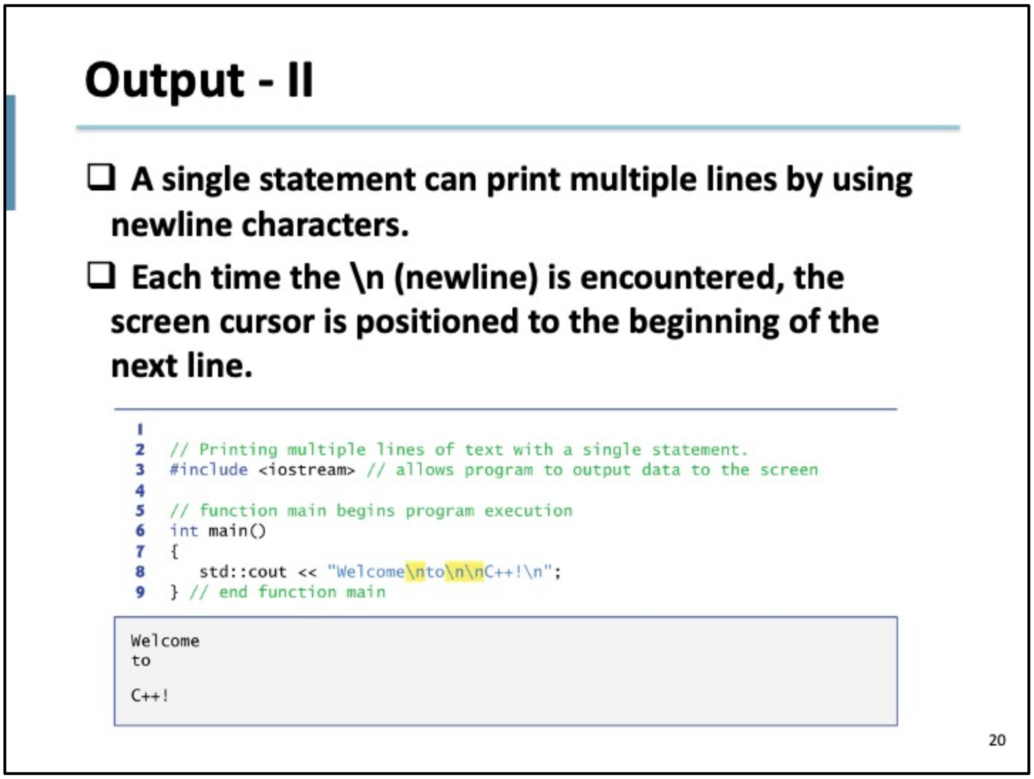
Let's look at our first C++ program that prints out a simple hello message. This is a C++ code. The first thing you need to do is to include that allows your program to use functions that are provided by the libraries, instead of writing your own. Including a library is like you go to HomeDepot to buy a driller so you can use the driller to do construction work instead of building a driller on your own which is very difficult.

All the program starts from a main function. This is where the operating system recognizes the entry pooint of your program. When you run your program, the first entry point you enter is the main block.

Inside the main function, we use std::cout to output an message. std::cout is a global object defined by the library for you to output message. The message under the quotation indicates it is a string. A string is a human readable text.

Another important message here is 'std::' represents the namespace. A namespace is where you can define your own things without naming conflict with others. Think about this ...

Finally, we return an integer from main function.



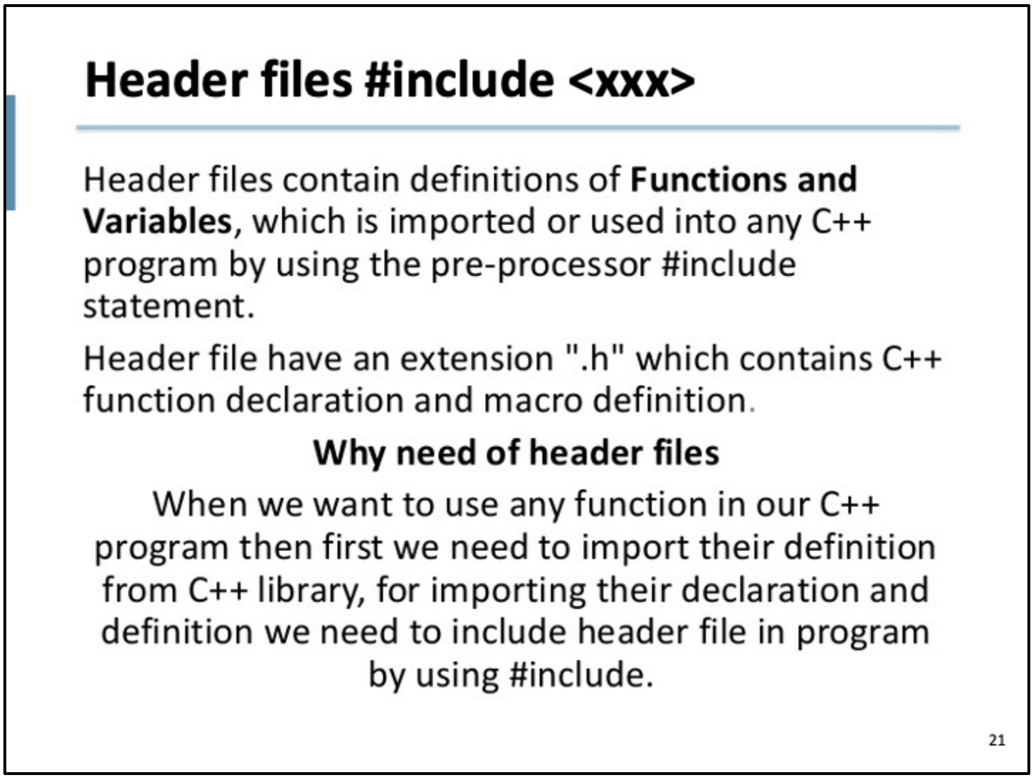
Output - II

- ❑ A single statement can print multiple lines by using newline characters.
- ❑ Each time the \n (newline) is encountered, the screen cursor is positioned to the beginning of the next line.

```cpp

// Printing multiple lines of text with a single statement.
#include <iostream> // allows program to output data to the screen

// function main begins program execution
int main()
{
   std::cout << "Welcome\nto\n\nC++!\n";
} // end function main
```

```
Welcome
to

C++!
```

For this prining statement using std::cout, a single state ...\



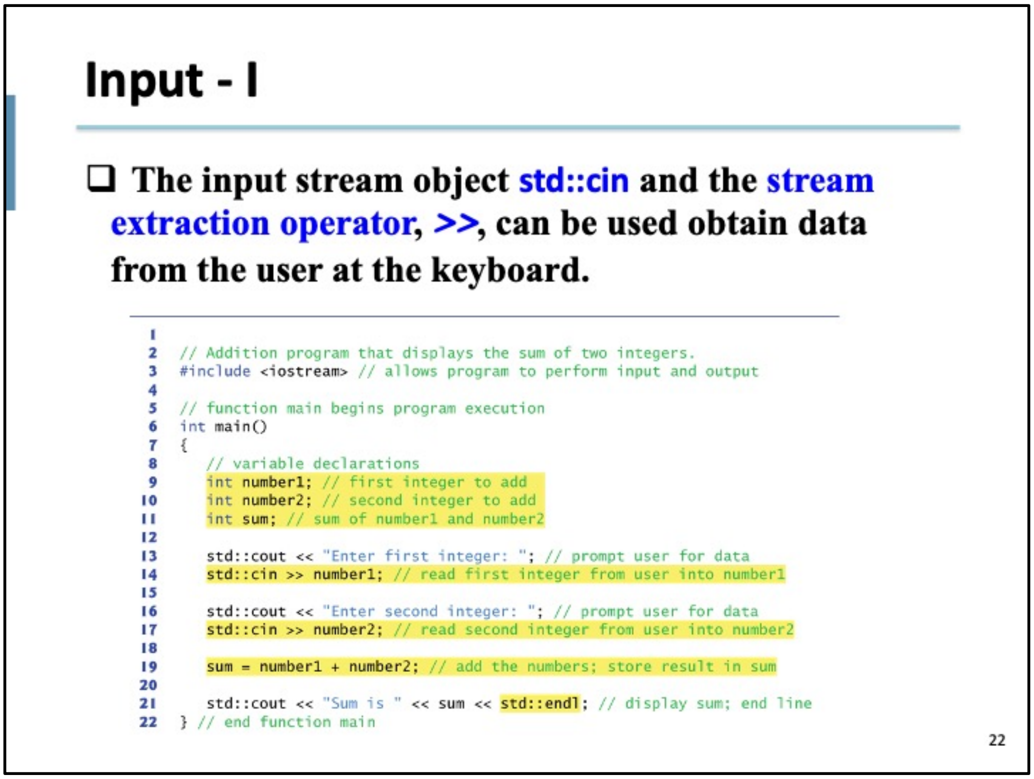
Header files #include <xxx>

Header files contain definitions of **Functions and Variables**, which is imported or used into any C++ program by using the pre-processor #include statement.

Header file have an extension ".h" which contains C++ function declaration and macro definition.

Why need of header files

When we want to use any function in our C++ program then first we need to import their definition from C++ library, for importing their declaration and definition we need to include header file in program by using #include.



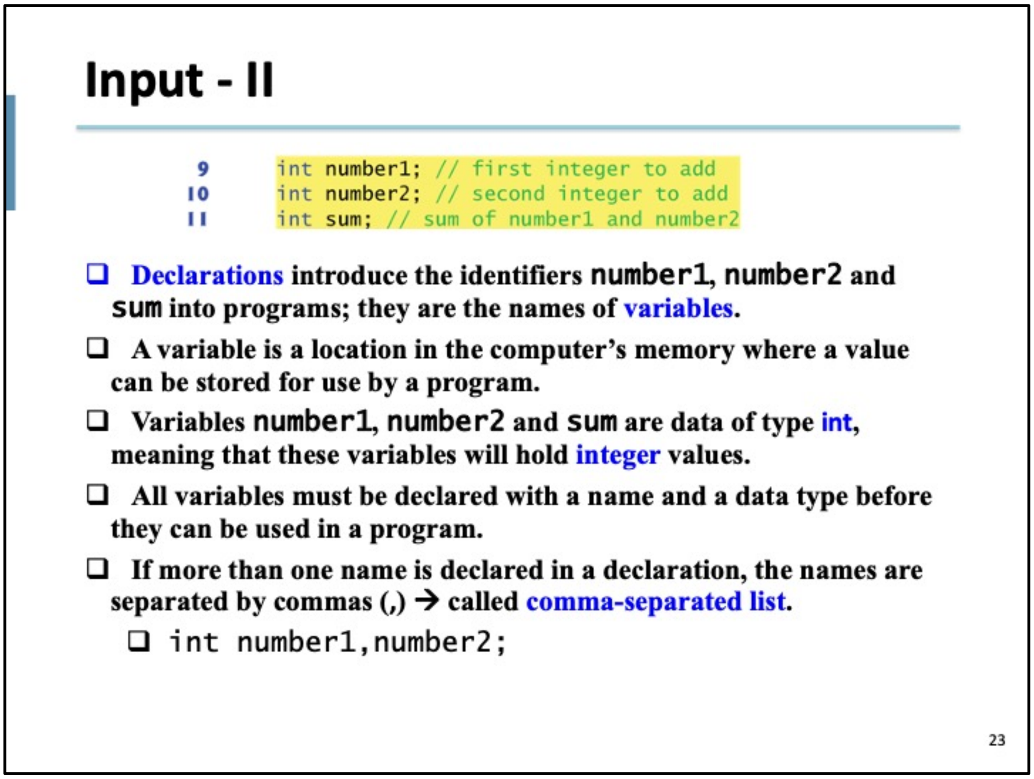
Input - I

- The input stream object **std::cin** and the **stream extraction operator**, **>>**, can be used obtain data from the user at the keyboard.

```

// Addition program that displays the sum of two integers.
#include <iostream> // allows program to perform input and output

// function main begins program execution
int main()
{
   // variable declarations
   int number1; // first integer to add
   int number2; // second integer to add
   int sum; // sum of number1 and number2

   std::cout << "Enter first integer: "; // prompt user for data
   std::cin >> number1; // read first integer from user into number1

   std::cout << "Enter second integer: "; // prompt user for data
   std::cin >> number2; // read second integer from user into number2

   sum = number1 + number2; // add the numbers; store result in sum

   std::cout << "Sum is " << sum << std::endl; // display sum; end line
} // end function main
```

Now we understand how to output a message. Another fundamental concept is input. We need to take data from users so we can do some processing on the data. In C++, the input stream ...

(explain number1, number2,...)

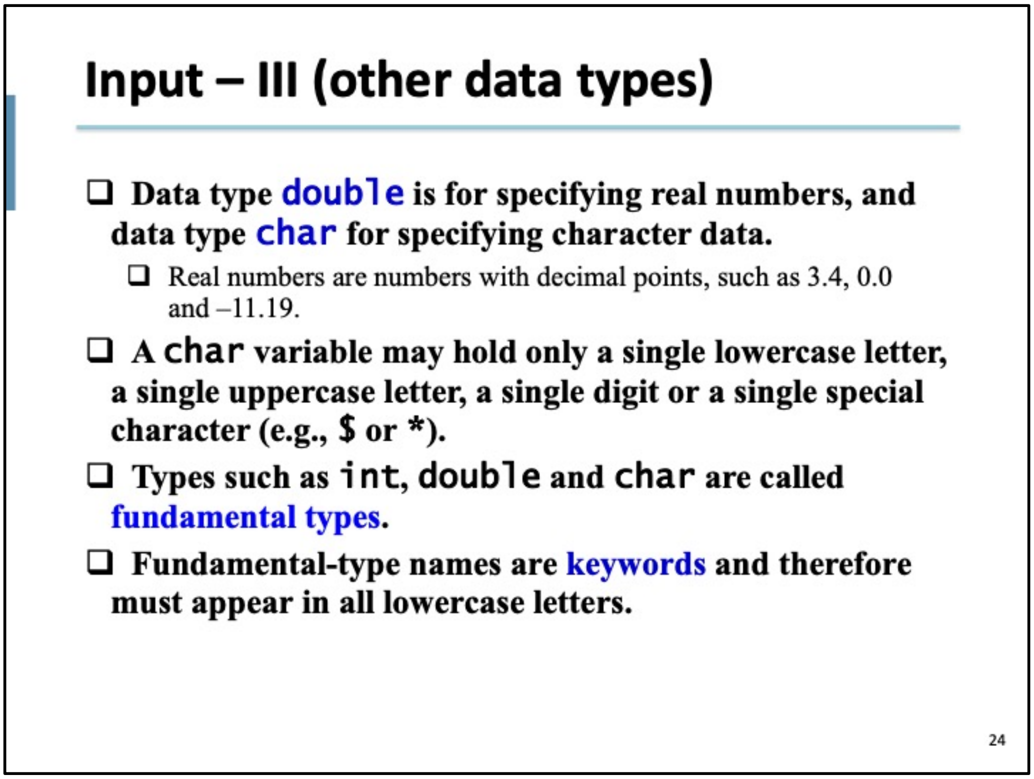


Input - II

```
int number1; // first integer to add
int number2; // second integer to add
int sum; // sum of number1 and number2
```

- ❑ **Declarations introduce the identifiers `number1`, `number2` and `sum` into programs; they are the names of variables.**
- ❑ **A variable is a location in the computer’s memory where a value can be stored for use by a program.**
- ❑ **Variables `number1`, `number2` and `sum` are data of type int, meaning that these variables will hold integer values.**
- ❑ **All variables must be declared with a name and a data type before they can be used in a program.**
- ❑ **If more than one name is declared in a declaration, the names are separated by commas (,) → called comma-separated list.**
  - ❑ `int number1,number2;`

Speaking of input, we must let the program know what data type your input is, right? Remember earlier in the slide I menion there are multiple data types, numbers, texts, binary, and so on. In our example, declaration ...

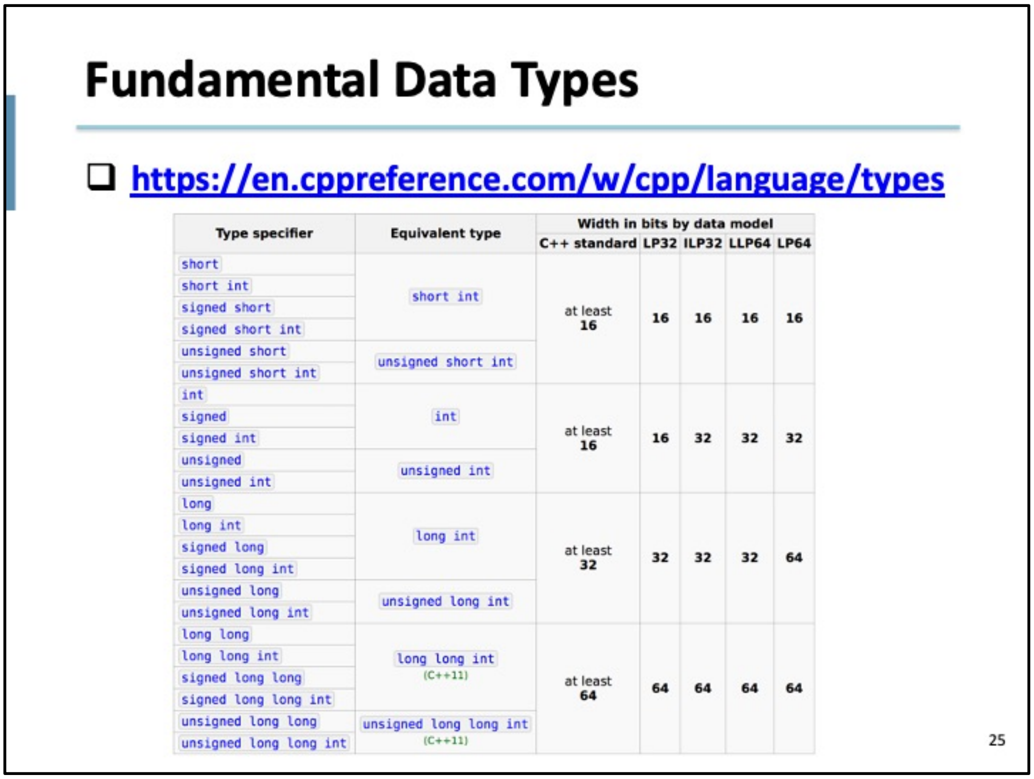


Input – III (other data types)

- ❑ Data type `double` is for specifying real numbers, and data type `char` for specifying character data.
  - ❑ Real numbers are numbers with decimal points, such as 3.4, 0.0 and –11.19.
- ❑ A `char` variable may hold only a single lowercase letter, a single uppercase letter, a single digit or a single special character (e.g., $ or *).
- ❑ Types such as `int`, `double` and `char` are called fundamental types.
- ❑ Fundamental-type names are keywords and therefore must appear in all lowercase letters.

24

There are other data types defined by the C++ language. For example, ...



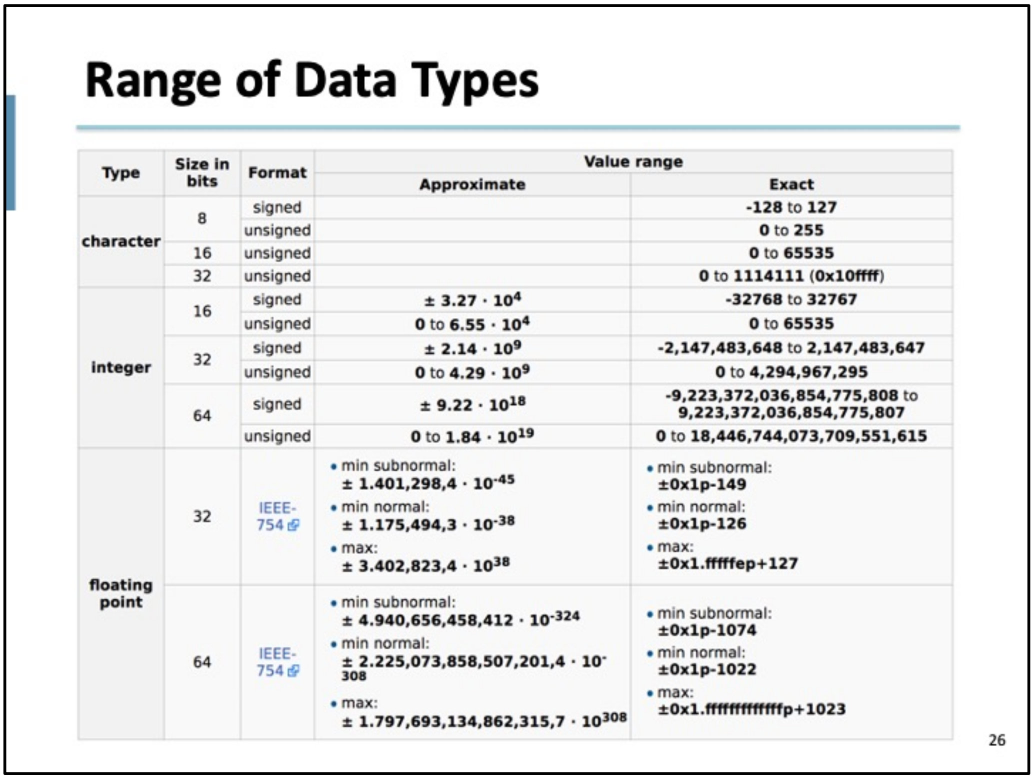
Fundamental Data Types

- https://en.cppreference.com/w/cpp/language/types

Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
short	short int	at least 16	16	16	16	16
short int						
signed short						
signed short int						
unsigned short	unsigned short int					
unsigned short int						
int	int	at least 16	16	32	32	32
signed						
signed int						
unsigned	unsigned int					
unsigned int						
long	long int	at least 32	32	32	32	64
long int						
signed long						
signed long int						
unsigned long	unsigned long int					
unsigned long int						
long long	long long int (C++11)	at least 64	64	64	64	64
long long int						
signed long long						
signed long long int						
unsigned long long	unsigned long long int (C++11)					
unsigned long long int						

This slides shows all the fundamental data types of C++. Each data type represents a particular number in inters or numerics. Each data type has a fixed width bit, where each bit represents a zero or one in the binary system of computer.

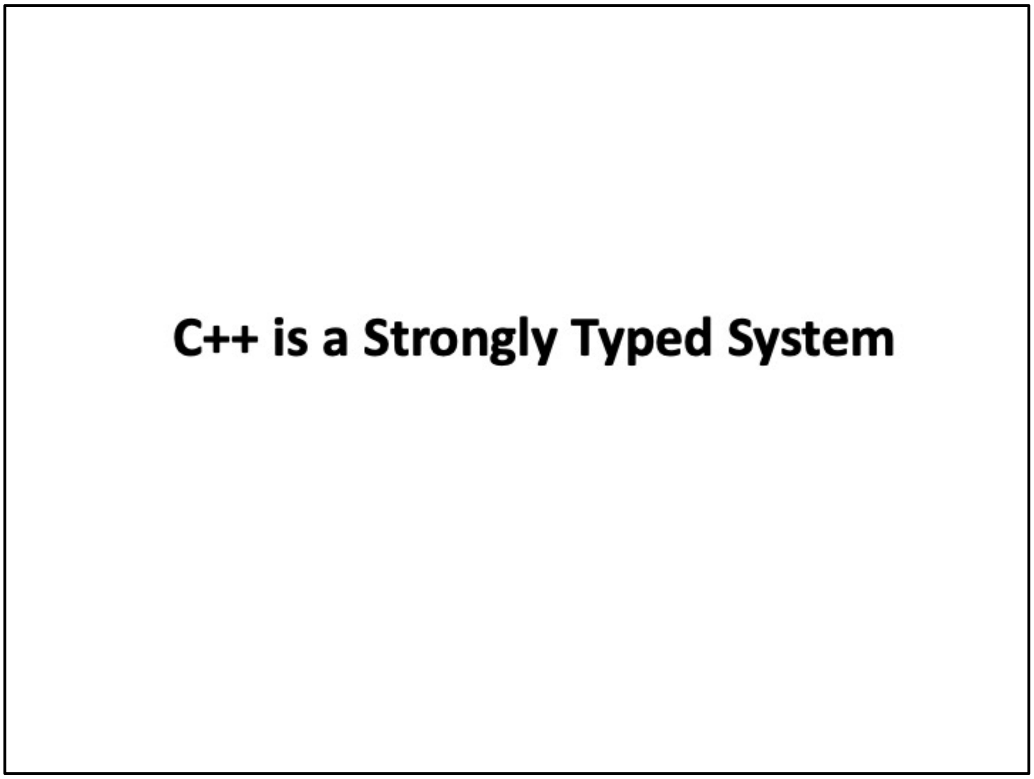
For example, if you define int, it has 32 bit; if you define double, it has 64 bit. The longer the bit width, the larger the value it can represent.



Range of Data Types

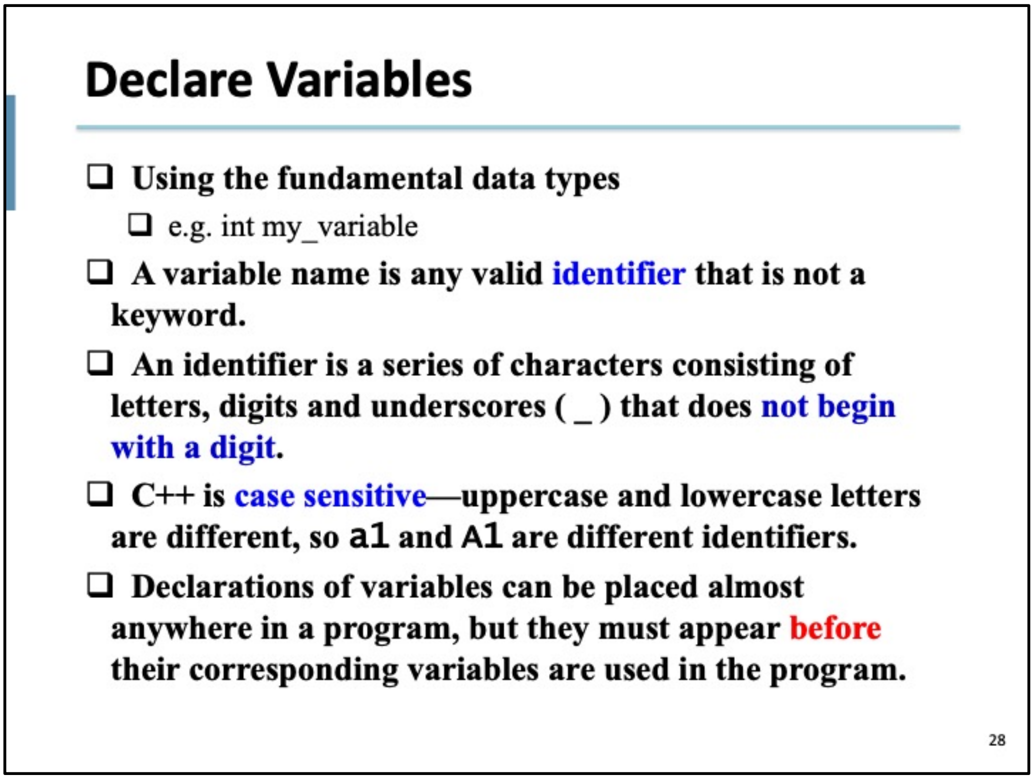
Type	Size in bits	Format	Value range	
			Approximate	Exact
character	8	signed		-128 to 127
		unsigned		0 to 255
	16	unsigned		0 to 65535
	32	unsigned		0 to 1114111 (0x10ffff)
integer	16	signed	± 3.27 · 10^4	-32768 to 32767
		unsigned	0 to 6.55 · 10^4	0 to 65535
	32	signed	± 2.14 · 10^9	-2,147,483,648 to 2,147,483,647
		unsigned	0 to 4.29 · 10^9	0 to 4,294,967,295
	64	signed	± 9.22 · 10^{18}	-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	• min subnormal: ± 1.401,298,4 · 10^{-45} • min normal: ± 1.175,494,3 · 10^{-38} • max: ± 3.402,823,4 · 10^{38}	• min subnormal: ±0x1p-149 • min normal: ±0x1p-126 • max: ±0x1.fffffep+127
	64	IEEE-754	• 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}	• min subnormal: ±0x1p-1074 • min normal: ±0x1p-1022 • max: ±0x1.fffffffffffffp+1023

Here is the slide that summarizes the range of data types.



C++ is a Strongly Typed System

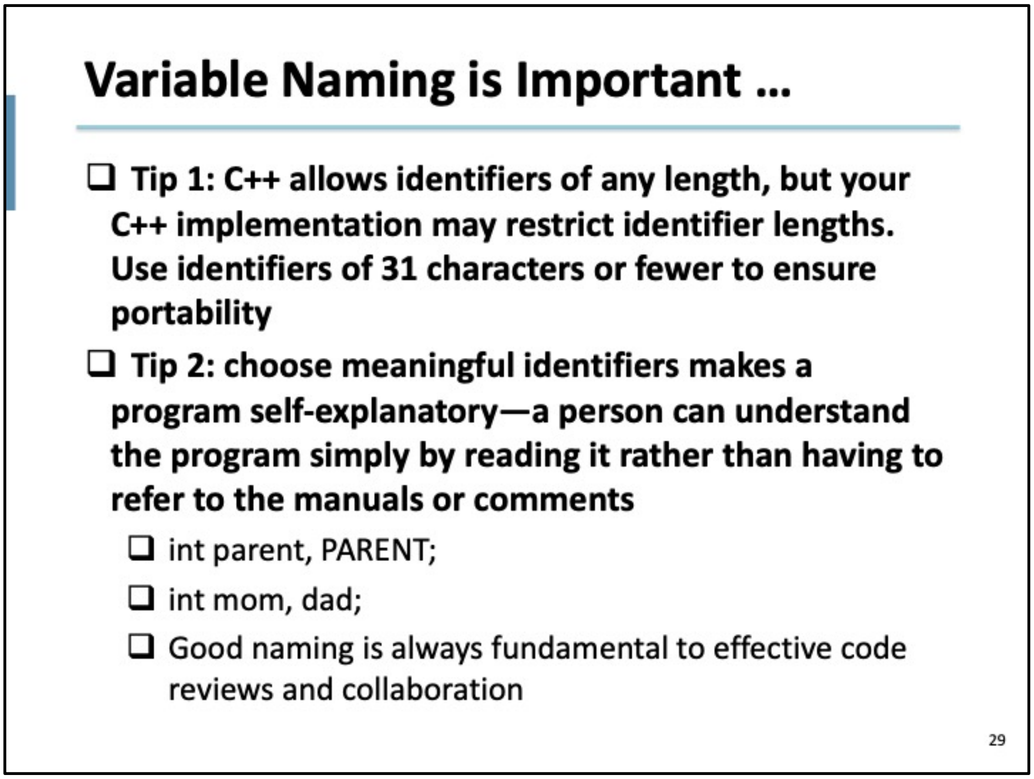
A key takeaway - C++ is a strongly typed system. All the variables you create must have a type. Each type has a predefined memory layout that the compiler can leverage to optimize the program.



Declare Variables

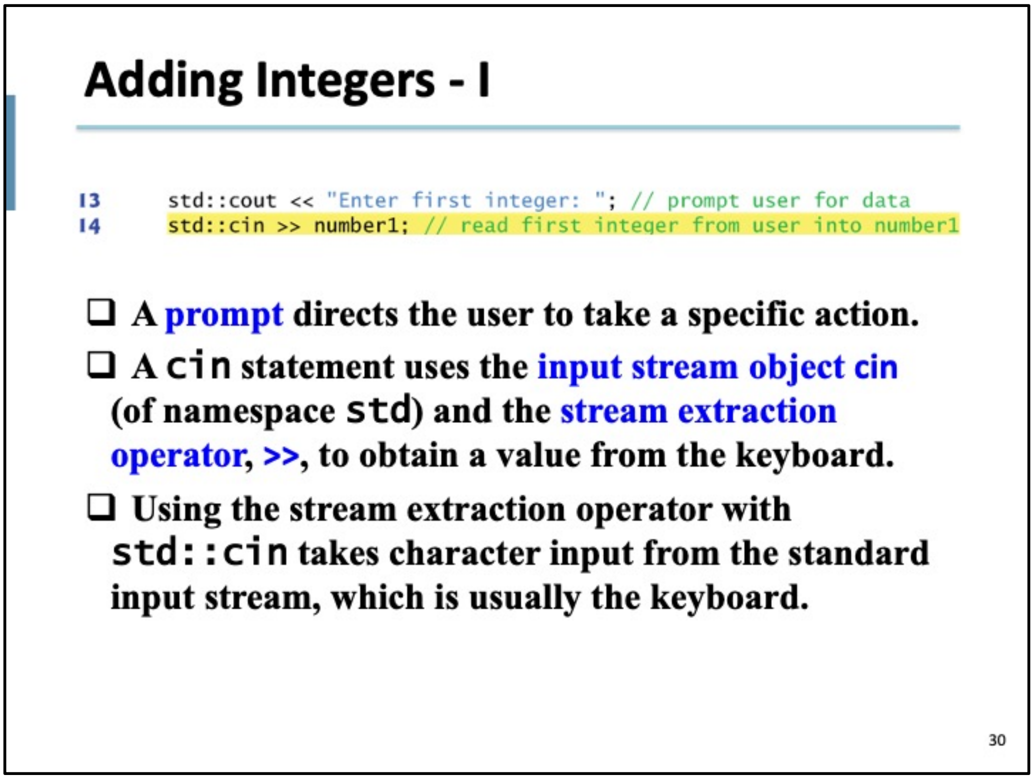
- ❑ **Using the fundamental data types**
  - ❑ e.g. int my_variable
- ❑ **A variable name is any valid identifier that is not a keyword.**
- ❑ **An identifier is a series of characters consisting of letters, digits and underscores (_) that does not begin with a digit.**
- ❑ **C++ is case sensitive—uppercase and lowercase letters are different, so `a1` and `A1` are different identifiers.**
- ❑ **Declarations of variables can be placed almost anywhere in a program, but they must appear before their corresponding variables are used in the program.**

Now we know the data type, we use it to declare variables.



Variable Naming is Important …

- ❑ **Tip 1: C++ allows identifiers of any length, but your C++ implementation may restrict identifier lengths. Use identifiers of 31 characters or fewer to ensure portability**
- ❑ **Tip 2: choose meaningful identifiers makes a program self-explanatory—a person can understand the program simply by reading it rather than having to refer to the manuals or comments**
  - ❑ int parent, PARENT;
  - ❑ int mom, dad;
  - ❑ Good naming is always fundamental to effective code reviews and collaboration



Adding Integers - I

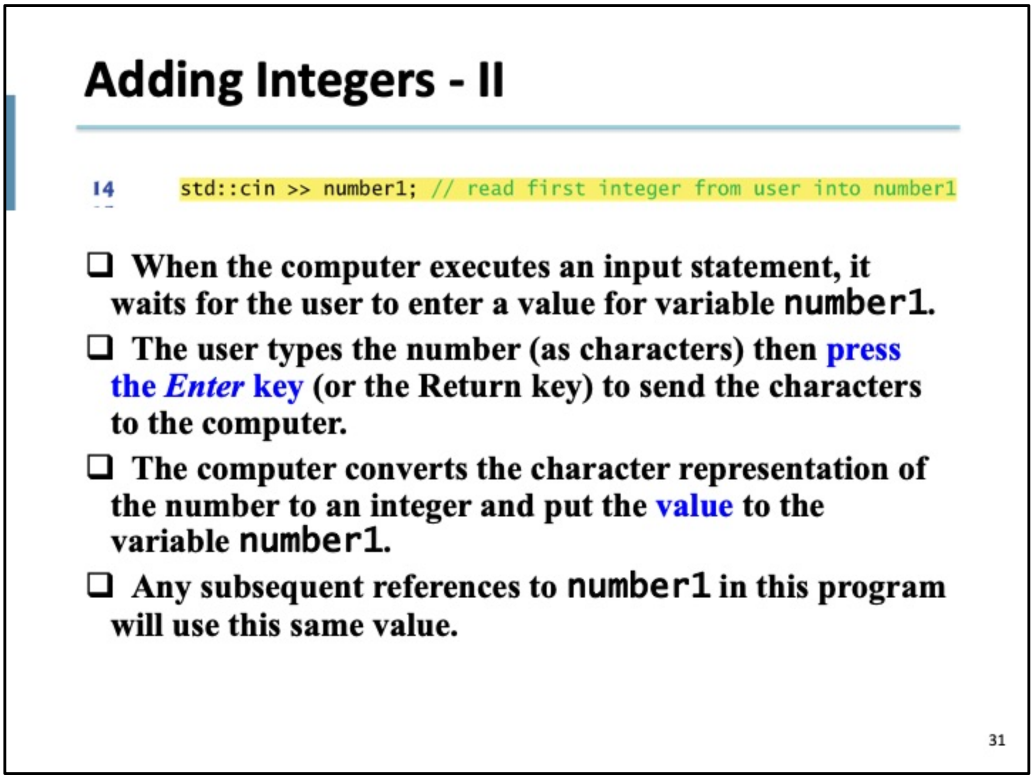
```
    std::cout << "Enter first integer: "; // prompt user for data
    std::cin >> number1; // read first integer from user into number1
```

□ A **prompt** directs the user to take a specific action.

□ A `cin` statement uses the **input stream object cin** (of namespace `std`) and the **stream extraction operator, >>**, to obtain a value from the keyboard.

□ Using the stream extraction operator with `std::cin` takes character input from the standard input stream, which is usually the keyboard.

Let's go back to our previous example of input. I have declared a variable "number1" and now I want to assign it a value from the user. Here I use std::cin, which is a global object defined by the C++ library for your program to digest data from the console. When the program encounters this instruction, a prompt ....

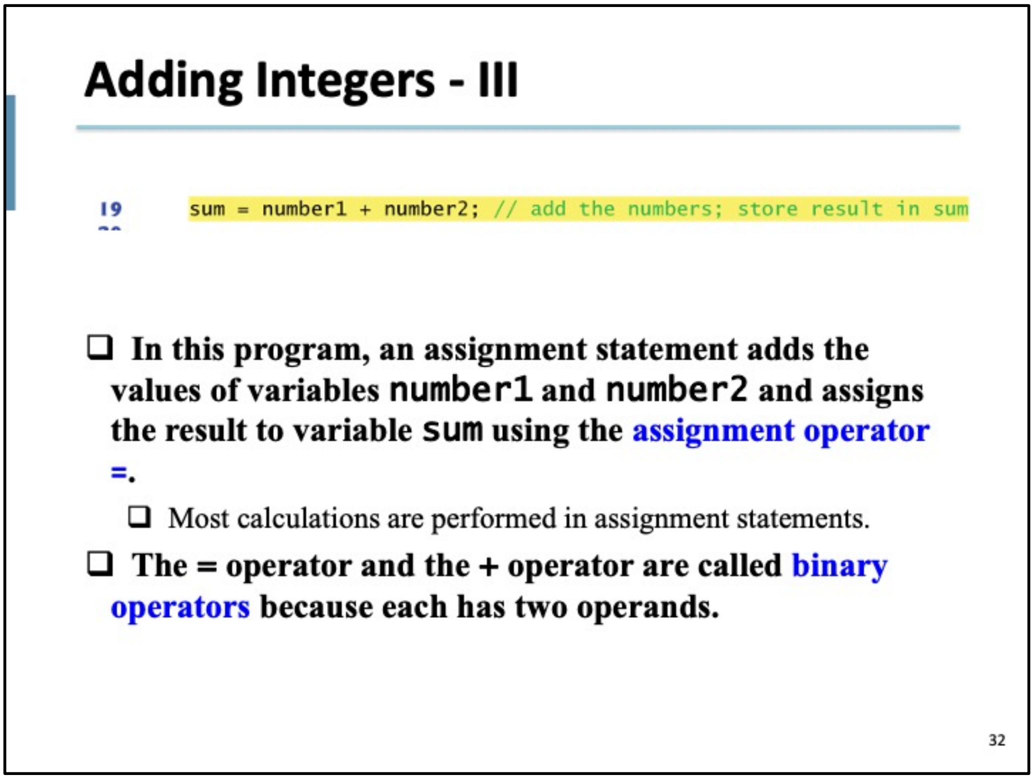


Adding Integers - II

```
std::cin >> number1; // read first integer from user into number1
```

- ❑ **When the computer executes an input statement, it waits for the user to enter a value for variable `number1`.**
- ❑ **The user types the number (as characters) then press the *Enter* key (or the Return key) to send the characters to the computer.**
- ❑ **The computer converts the character representation of the number to an integer and put the value to the variable `number1`.**
- ❑ **Any subsequent references to `number1` in this program will use this same value.**

When the program execution encounter the std::cin instruction, it waits for the user to enter ...



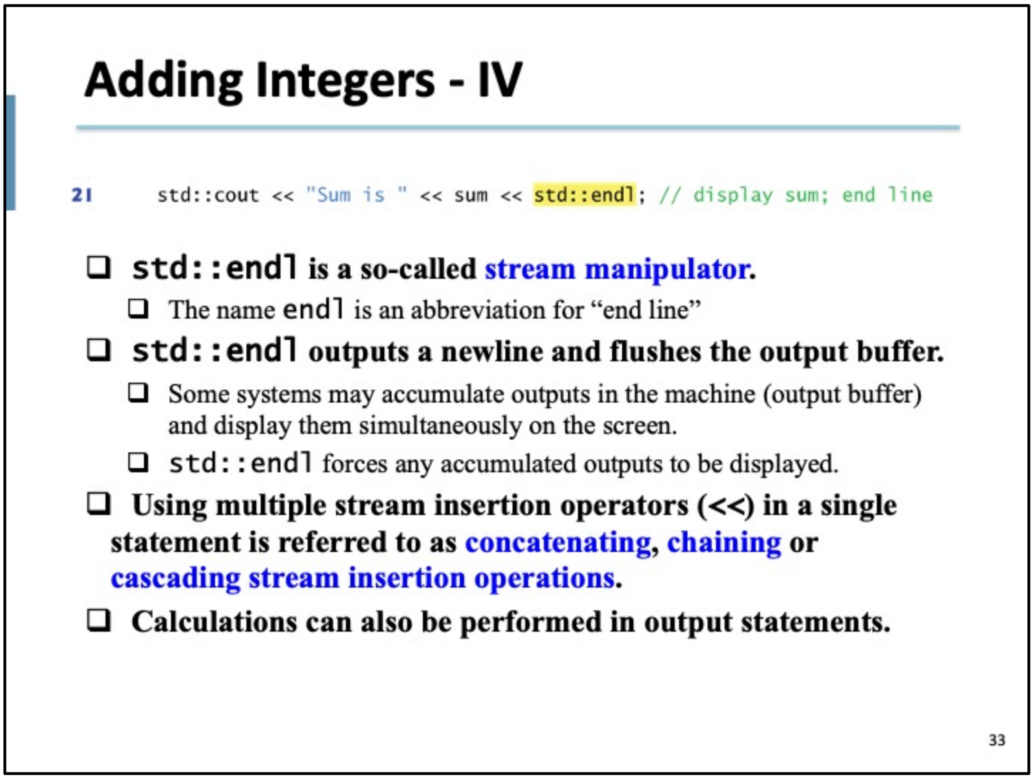
Adding Integers - III

```
sum = number1 + number2; // add the numbers; store result in sum
```

- ❑ In this program, an assignment statement adds the values of variables `number1` and `number2` and assigns the result to variable `sum` using the assignment operator =.
  - ❑ Most calculations are performed in assignment statements.
- ❑ The = operator and the + operator are called binary operators because each has two operands.

In this program ... using the assignment operator '='. Assignment operator is the most fundamental building block to construct an expression. A program consists of multiple expressions to describe your behavior.

Notice that the equal sign "=" has nothing to do with the equation you learn from high school, for example, you may need to solve an equation y equals x square plus 1. That is an equation. In C++ program, this is an assignment. The equal sign is an assignment operator and the + is a binary operator that adds two number and return the result to the left-hand side.

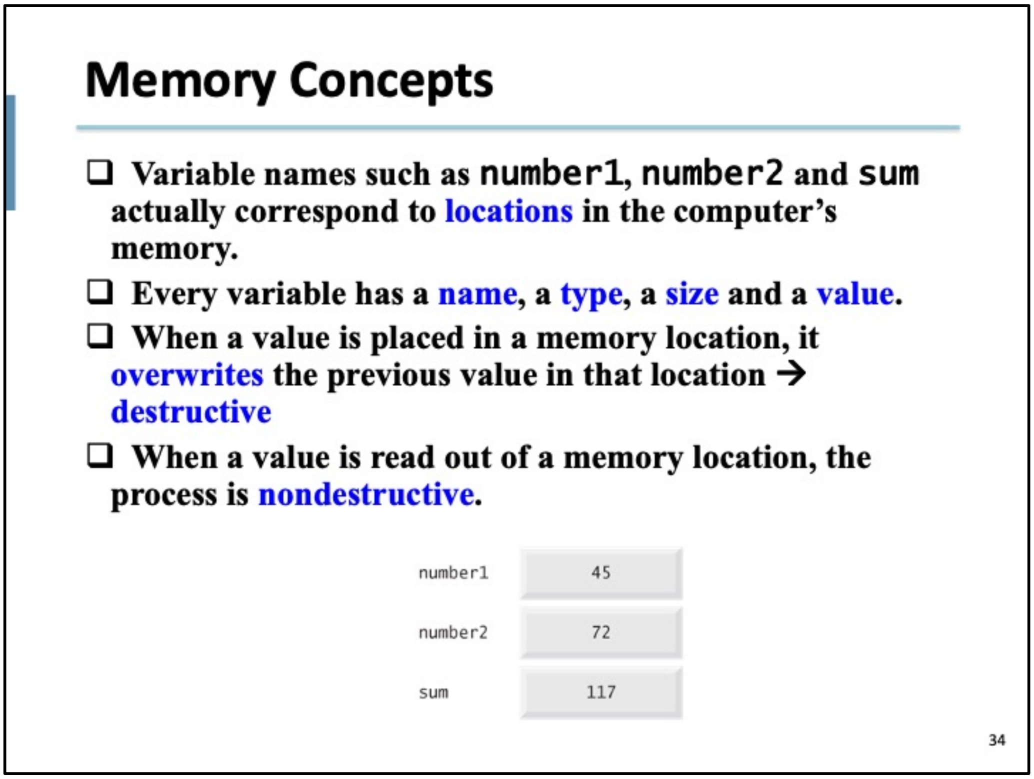


Adding Integers - IV

```
std::cout << "Sum is " << sum << std::endl; // display sum; end line
```

- ❑ **`std::endl` is a so-called stream manipulator.**
  - ❑ The name `endl` is an abbreviation for “end line”
- ❑ **`std::endl` outputs a newline and flushes the output buffer.**
  - ❑ Some systems may accumulate outputs in the machine (output buffer) and display them simultaneously on the screen.
  - ❑ `std::endl` forces any accumulated outputs to be displayed.
- ❑ **Using multiple stream insertion operators (<<) in a single statement is referred to as concatenating, chaining or cascading stream insertion operations.**
- ❑ **Calculations can also be performed in output statements.**

Once we complete this simple addition, we can cout the value of sum. Here I use std::endl to break the line. std::endl is a stream manipulator that outputs a new line and flushes the output stored in std::cout.



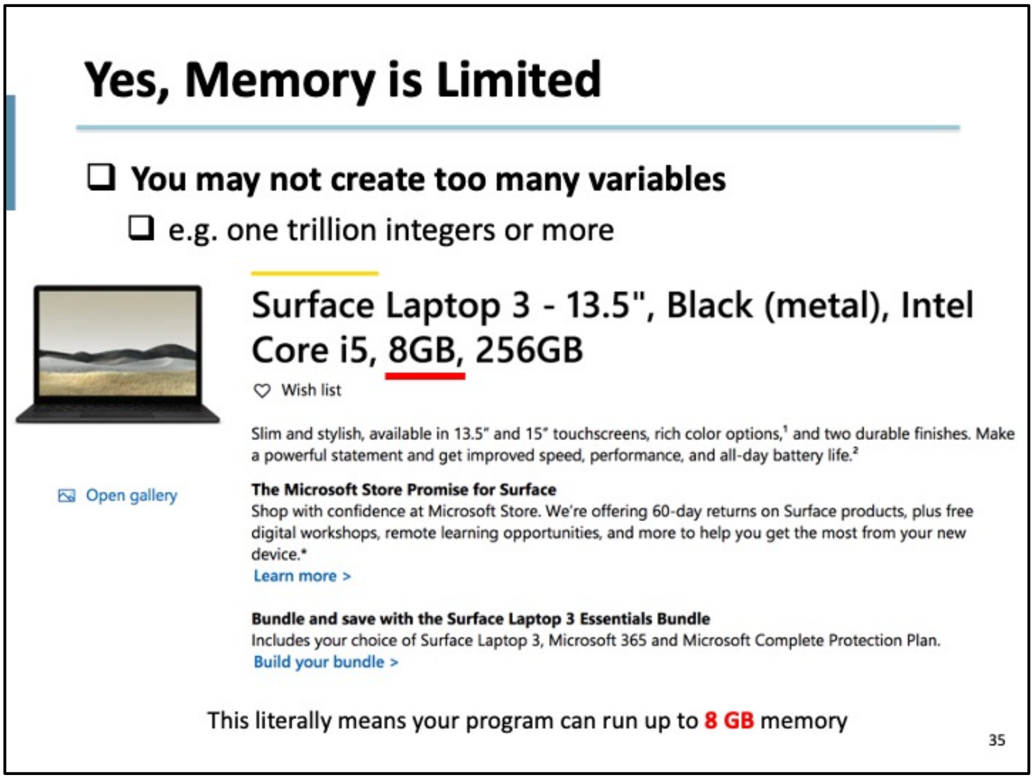
Memory Concepts

- ❑ Variable names such as `number1`, `number2` and `sum` actually correspond to **locations** in the computer's memory.
- ❑ Every variable has a **name**, a **type**, a **size** and a **value**.
- ❑ When a value is placed in a memory location, it **overwrites** the previous value in that location → **destructive**
- ❑ When a value is read out of a memory location, the process is **nondestructive**.

number1	45
number2	72
sum	117

Now we have seen a basic C++ program that takes two input number1 and number2 from the user, sum the two numbers, and store the result in a variable. It's important for us to know how this program actually works on hardware, or memory. Remember that in the first lecture, we talked about C++ is powerufl in mapping your program directly into hardware. One of the fundamental concepts is memory architecture of your data.

(explain each item ...)



Yes, Memory is Limited

- ❑ **You may not create too many variables**
  - ❑ e.g. one trillion integers or more

Surface Laptop 3 - 13.5", Black (metal), Intel Core i5, 8GB, 256GB

Slim and stylish, available in 13.5" and 15" touchscreens, rich color options,[1] and two durable finishes. Make a powerful statement and get improved speed, performance, and all-day battery life.[2]

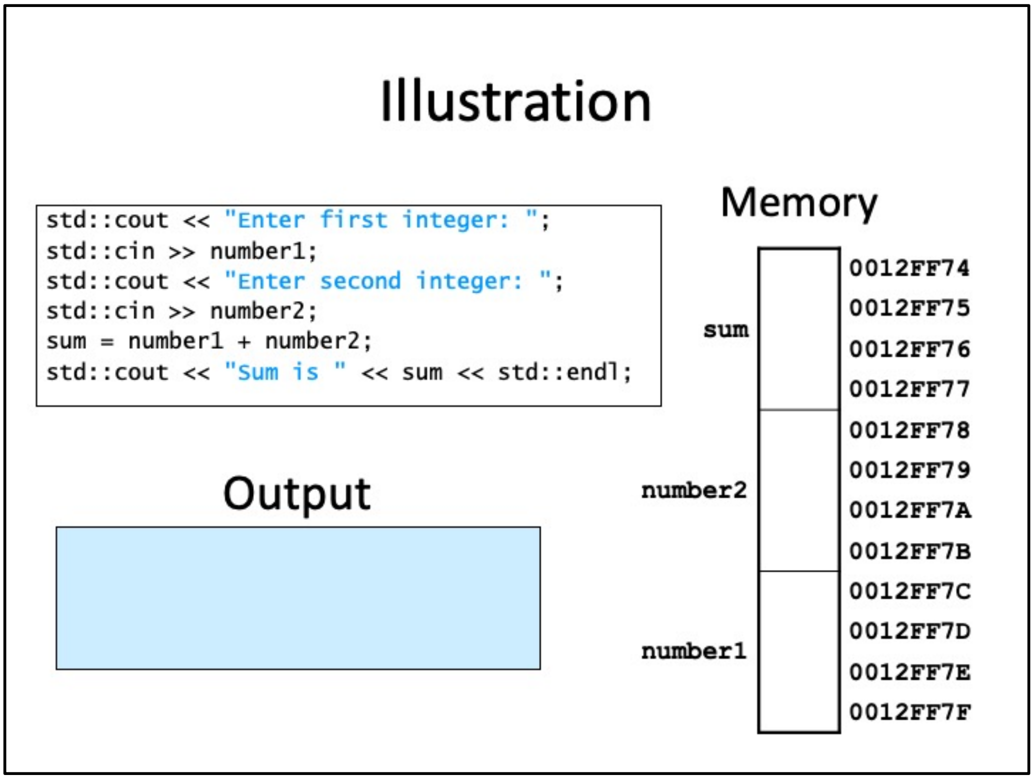
The Microsoft Store Promise for Surface
Shop with confidence at Microsoft Store. We're offering 60-day returns on Surface products, plus free digital workshops, remote learning opportunities, and more to help you get the most from your new device.*

Bundle and save with the Surface Laptop 3 Essentials Bundle
Includes your choice of Surface Laptop 3, Microsoft 365 and Microsoft Complete Protection Plan.

This literally means your program can run up to **8 GB** memory

35

You probably already know, memory is limited. Yes, that means you may not create too many variables, for example, a trillion integers or more. The largest size of the program you can run on your machine depends on the memory size you have. You can get this information when you purchase a laptop or a desktop. For example, below I am showing you a 13 inch surface laptop that has 8GB memory. This literally means your program can run up to 8 GB memory.



Illustration

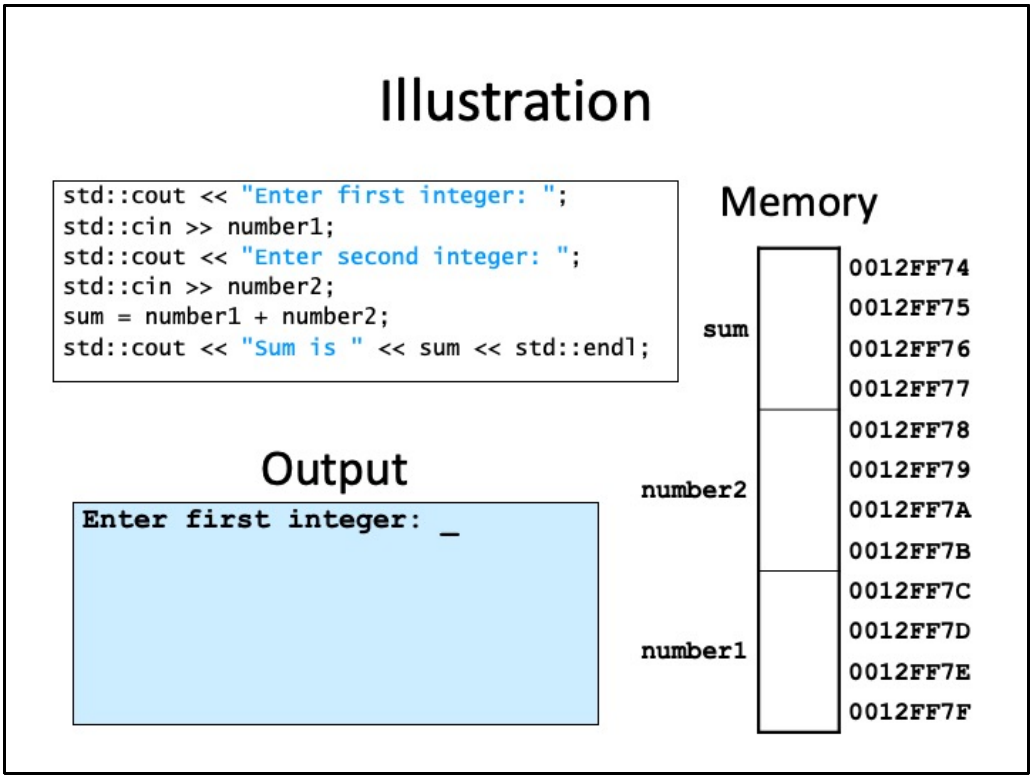
```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

Output

Memory

To give you a better idea about how this simple program works on hardware, let's illustrate the runtime. Here I have the program that takes two inputs and sums their values to an output. On the right-hand side, I have the memory with variable name on the left and their virtual address from byte to byte on the right side. Remember each integer is indeed a 32-bit data, that is 4 bytes, because each byte has 8 bits.

Let's run the program.



Illustration

```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

Memory

sum	0012FF74 0012FF75 0012FF76 0012FF77
number2	0012FF78 0012FF79 0012FF7A 0012FF7B
number1	0012FF7C 0012FF7D 0012FF7E 0012FF7F

Output

```
Enter first integer: _
```

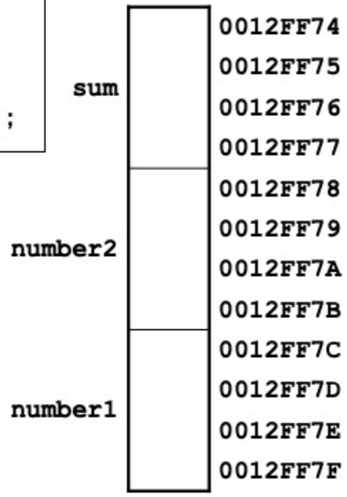
Illustration

```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

Output

```
Enter first integer: 45_
```

Memory

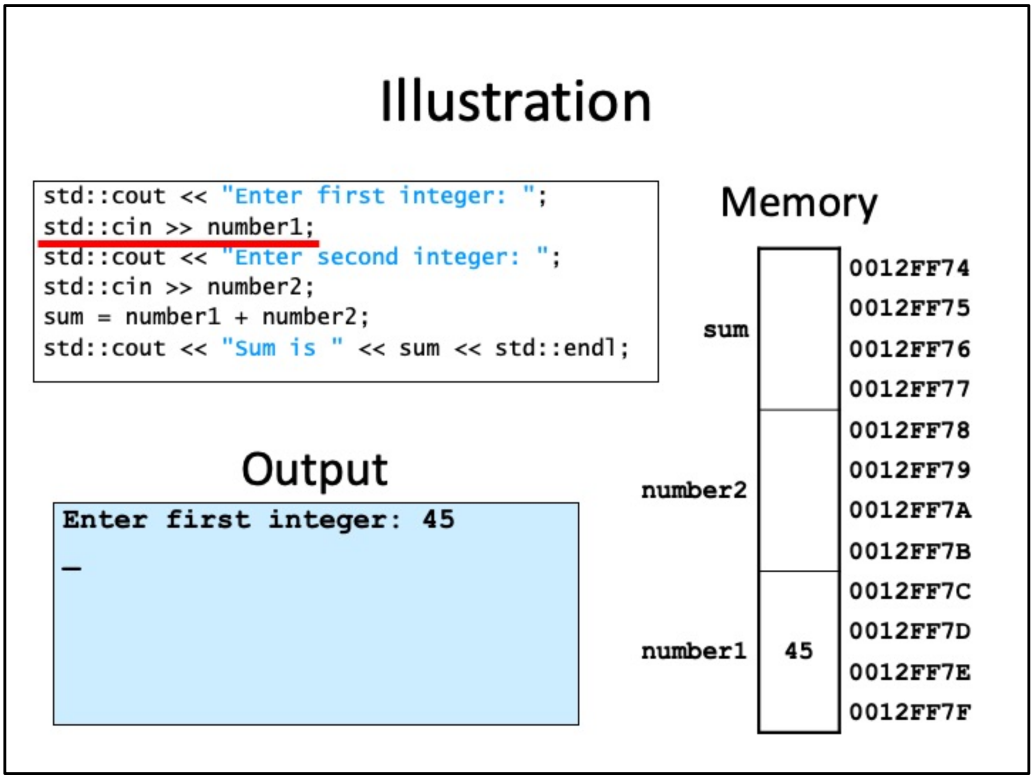


sum

number2

number1

0012FF74
0012FF75
0012FF76
0012FF77
0012FF78
0012FF79
0012FF7A
0012FF7B
0012FF7C
0012FF7D
0012FF7E
0012FF7F



Illustration

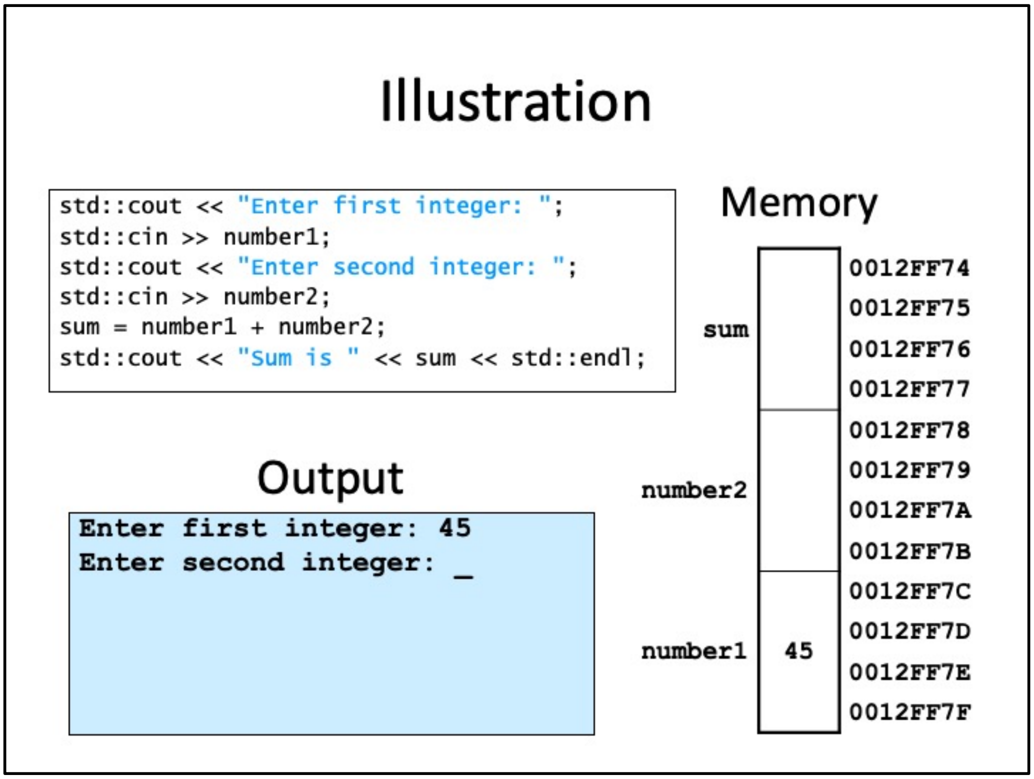
```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

Output

```
Enter first integer: 45
_
```

Memory

sum		0012FF74 0012FF75 0012FF76 0012FF77
number2		0012FF78 0012FF79 0012FF7A 0012FF7B
number1	45	0012FF7C 0012FF7D 0012FF7E 0012FF7F



Illustration

```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

Output

```
Enter first integer: 45
Enter second integer: _
```

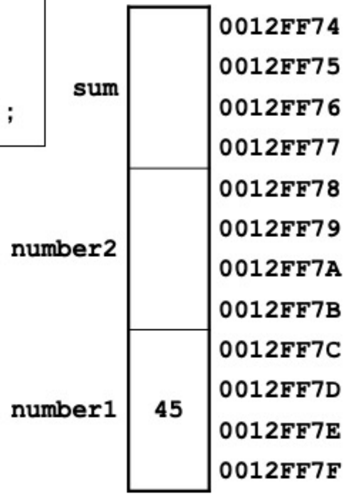
Memory

sum		0012FF74
		0012FF75
		0012FF76
		0012FF77
number2		0012FF78
		0012FF79
		0012FF7A
		0012FF7B
number1	45	0012FF7C
		0012FF7D
		0012FF7E
		0012FF7F

Illustration

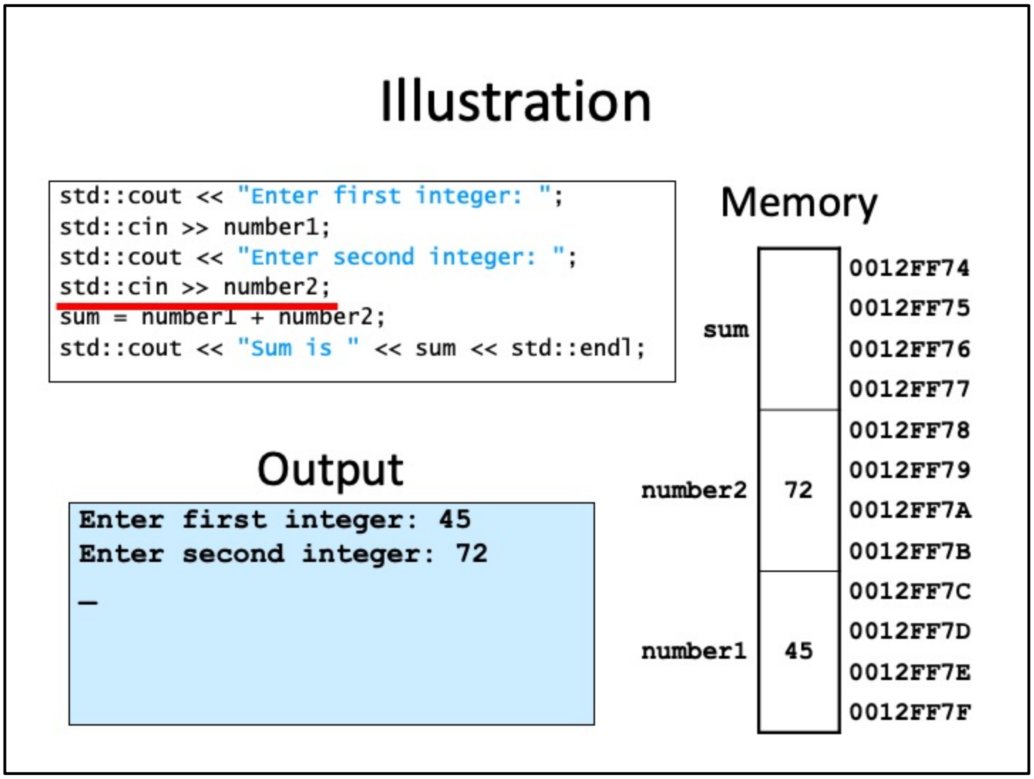
```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

Memory



Output

```
Enter first integer: 45
Enter second integer: 72_
```



Illustration

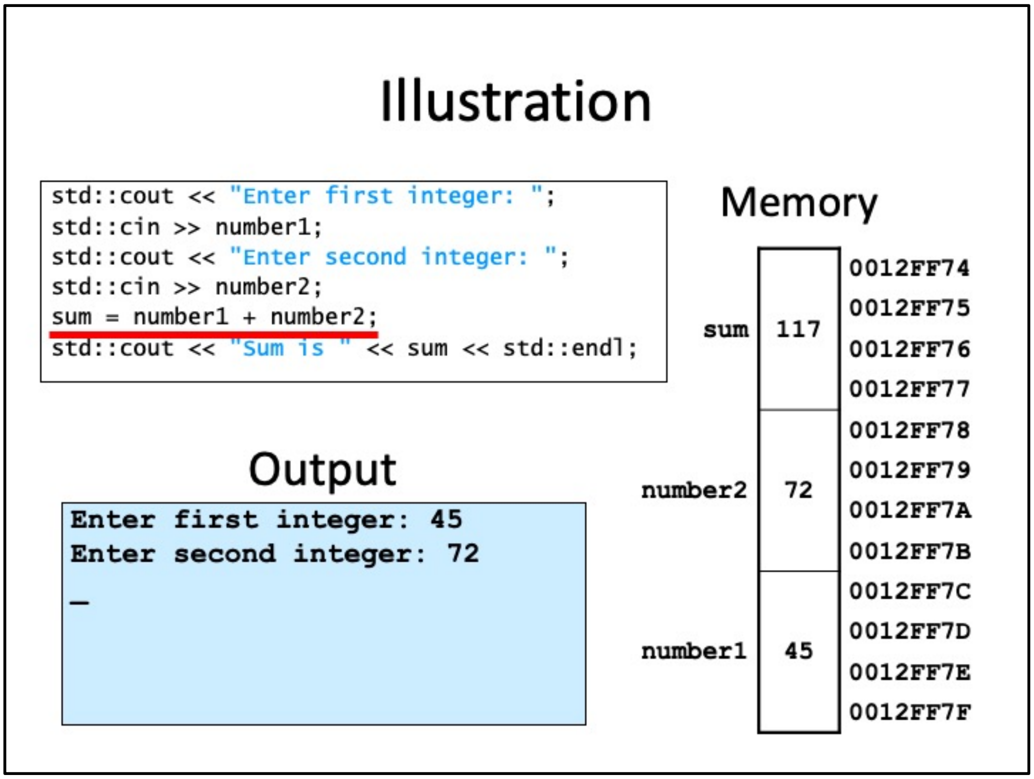
```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

Output

```
Enter first integer: 45
Enter second integer: 72

_
```

Memory

sum		0012FF74 0012FF75 0012FF76 0012FF77
number2	72	0012FF78 0012FF79 0012FF7A 0012FF7B
number1	45	0012FF7C 0012FF7D 0012FF7E 0012FF7F



Illustration

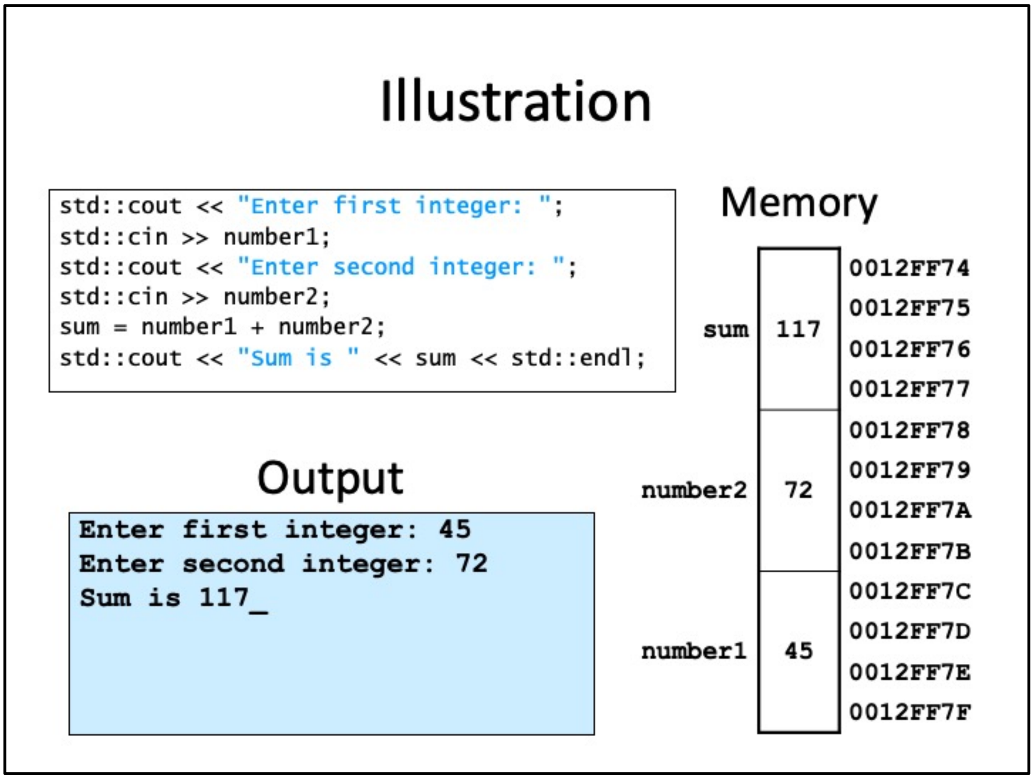
```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

Output

```
Enter first integer: 45
Enter second integer: 72

_
```

Memory

sum	117	0012FF74
		0012FF75
		0012FF76
		0012FF77
number2	72	0012FF78
		0012FF79
		0012FF7A
		0012FF7B
number1	45	0012FF7C
		0012FF7D
		0012FF7E
		0012FF7F



Illustration

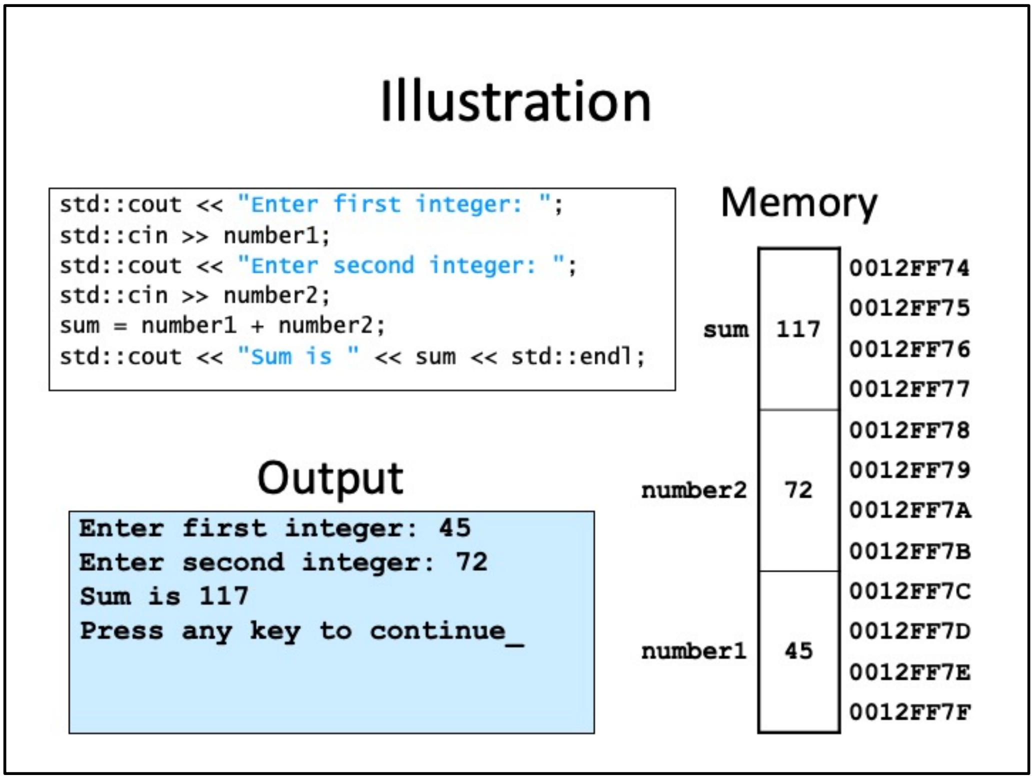
```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

Output

```
Enter first integer: 45
Enter second integer: 72
Sum is 117_
```

Memory

sum	117	0012FF74 0012FF75 0012FF76 0012FF77
number2	72	0012FF78 0012FF79 0012FF7A 0012FF7B
number1	45	0012FF7C 0012FF7D 0012FF7E 0012FF7F



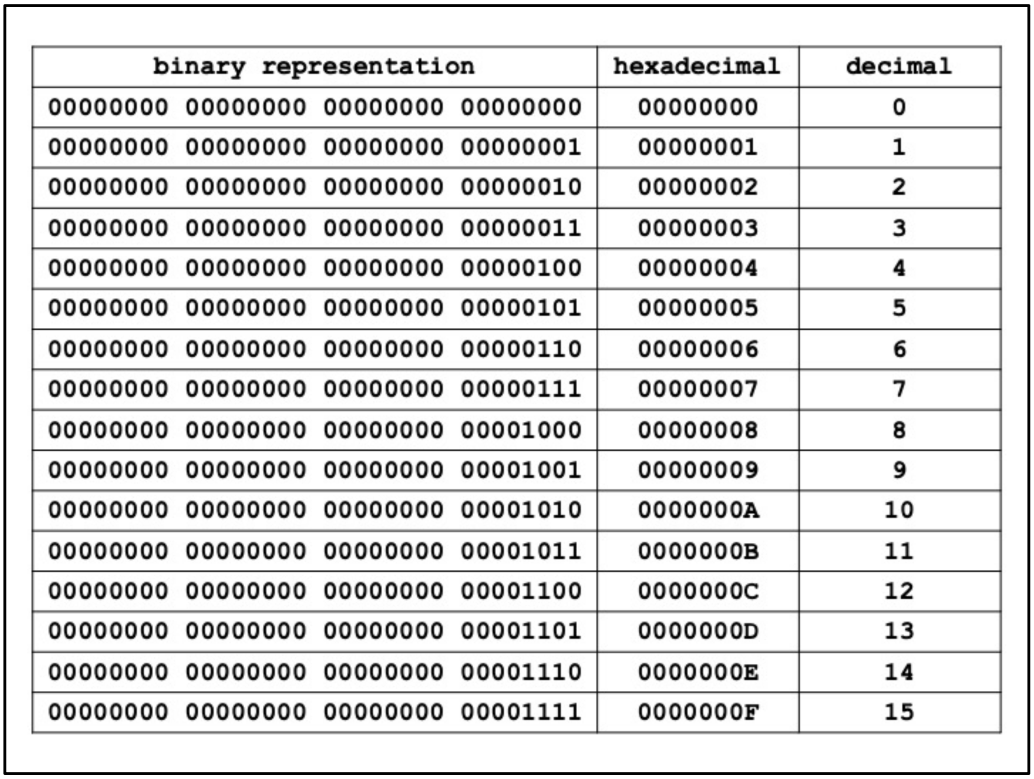
Illustration

```
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout << "Sum is " << sum << std::endl;
```

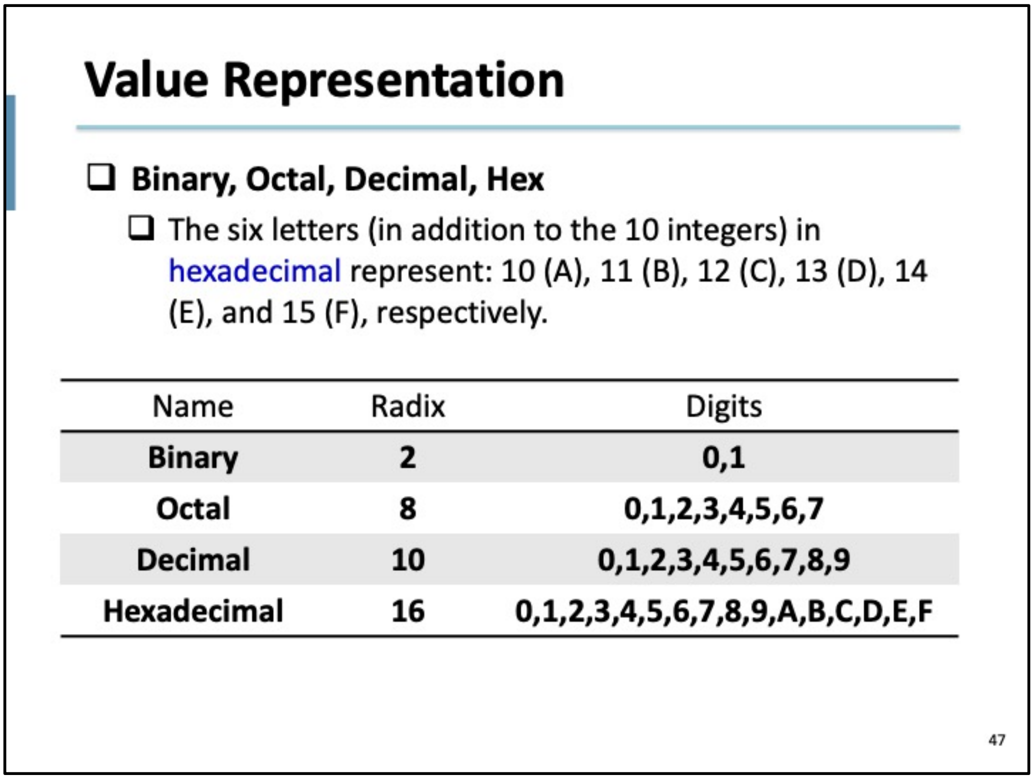
Output

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
Press any key to continue_
```

Memory



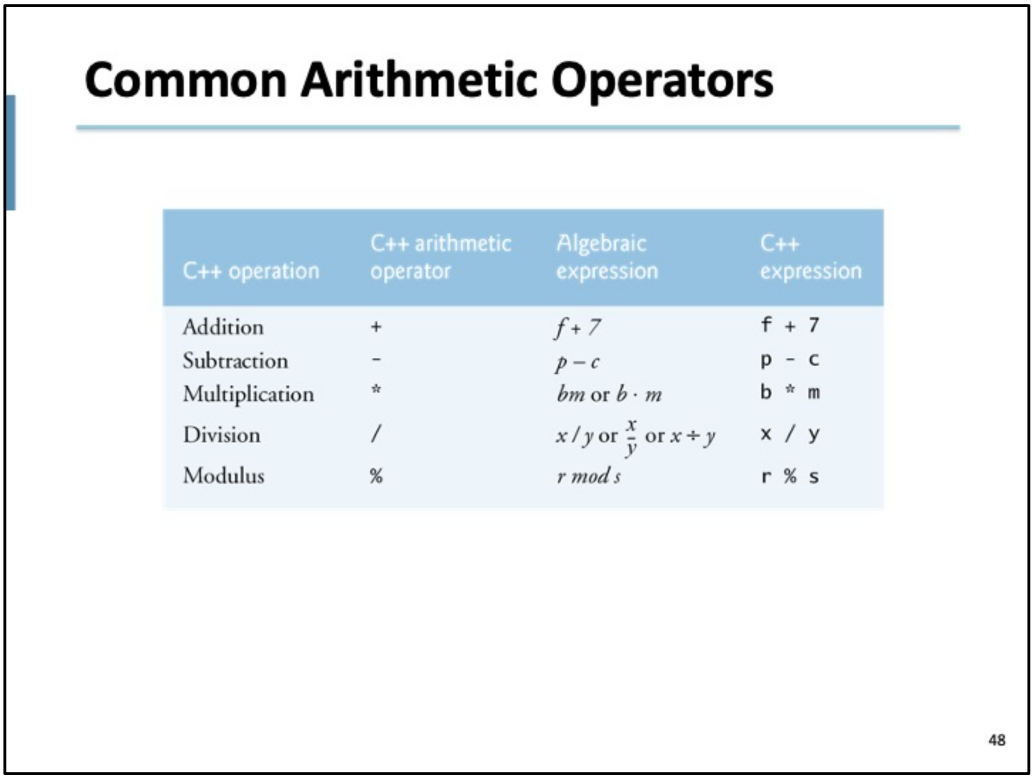
binary representation	hexadecimal	decimal
00000000 00000000 00000000 00000000	00000000	0
00000000 00000000 00000000 00000001	00000001	1
00000000 00000000 00000000 00000010	00000002	2
00000000 00000000 00000000 00000011	00000003	3
00000000 00000000 00000000 00000100	00000004	4
00000000 00000000 00000000 00000101	00000005	5
00000000 00000000 00000000 00000110	00000006	6
00000000 00000000 00000000 00000111	00000007	7
00000000 00000000 00000000 00001000	00000008	8
00000000 00000000 00000000 00001001	00000009	9
00000000 00000000 00000000 00001010	0000000A	10
00000000 00000000 00000000 00001011	0000000B	11
00000000 00000000 00000000 00001100	0000000C	12
00000000 00000000 00000000 00001101	0000000D	13
00000000 00000000 00000000 00001110	0000000E	14
00000000 00000000 00000000 00001111	0000000F	15



Value Representation

- ❑ **Binary, Octal, Decimal, Hex**
  - ❑ The six letters (in addition to the 10 integers) in hexadecimal represent: 10 (A), 11 (B), 12 (C), 13 (D), 14 (E), and 15 (F), respectively.

Name	Radix	Digits
Binary	**2**	**0,1**
Octal	**8**	**0,1,2,3,4,5,6,7**
Decimal	**10**	**0,1,2,3,4,5,6,7,8,9**
Hexadecimal	**16**	**0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F**

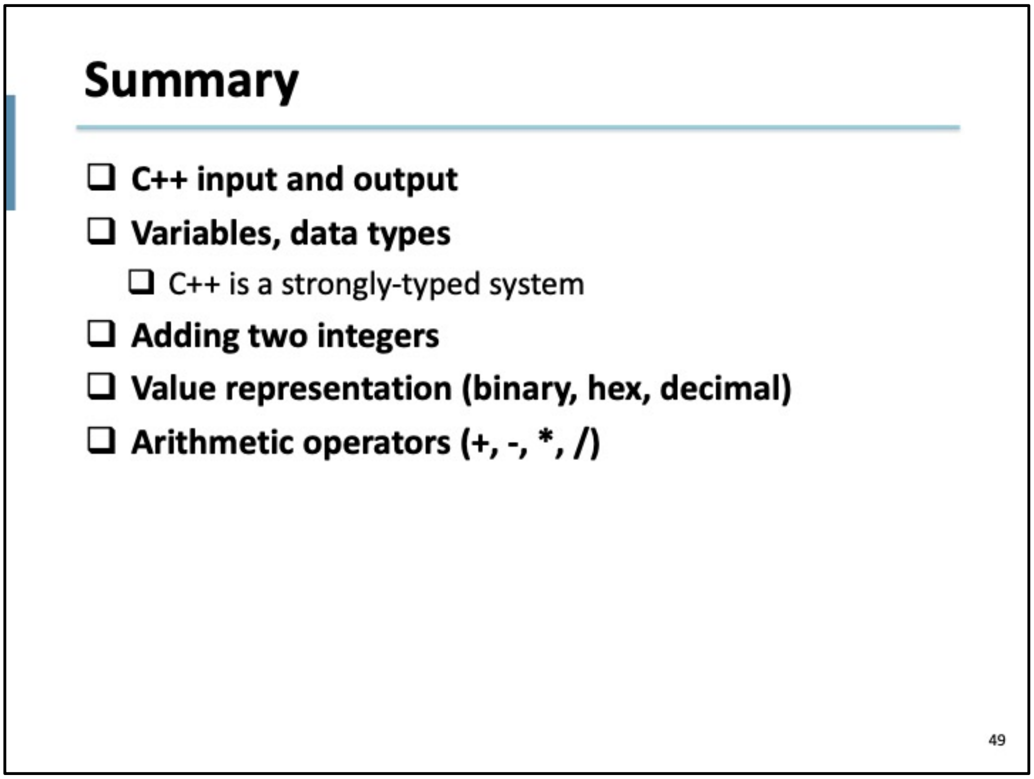


Common Arithmetic Operators

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm or $b \cdot m$	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Modulus	%	$r \ mod \ s$	`r % s`

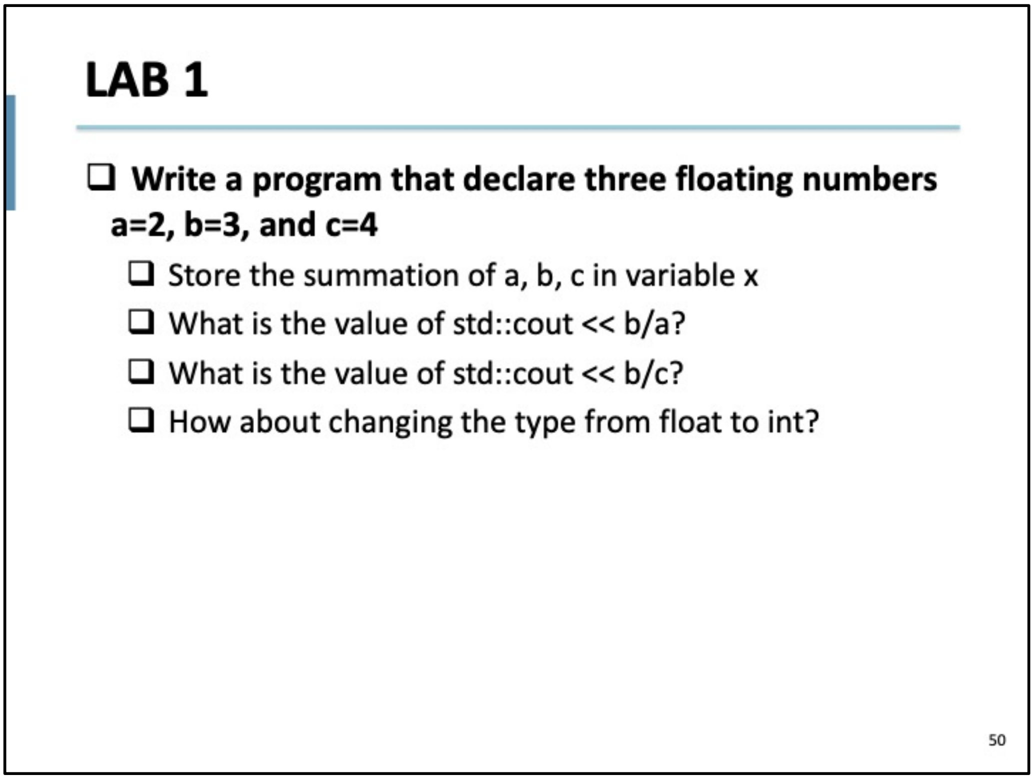
48

We talked about the addition operation. Indeed, there are many more. In C++, we have subtraction that subtracts a number from another, multiplication that multiples a number by another, division that divide a number by another, and modulus operation that finds the remainder of a division, for example, if you divide 7 by 3, the remainder is 1.



Summary

- □ **C++ input and output**
- □ **Variables, data types**
  - □ C++ is a strongly-typed system
- □ **Adding two integers**
- □ **Value representation (binary, hex, decimal)**
- □ **Arithmetic operators (+, -, *, /)**



LAB 1

- □ **Write a program that declare three floating numbers a=2, b=3, and c=4**
  - □ Store the summation of a, b, c in variable x
  - □ What is the value of std::cout << b/a?
  - □ What is the value of std::cout << b/c?
  - □ How about changing the type from float to int?