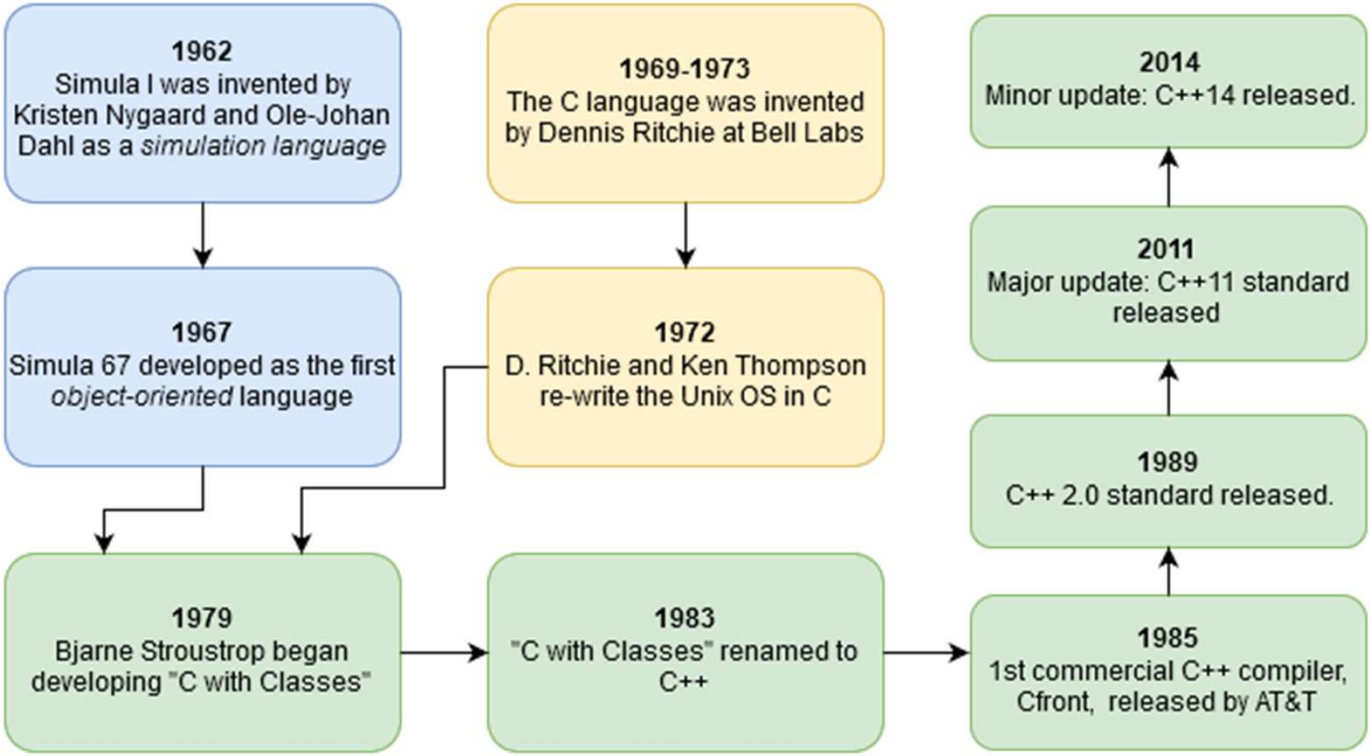
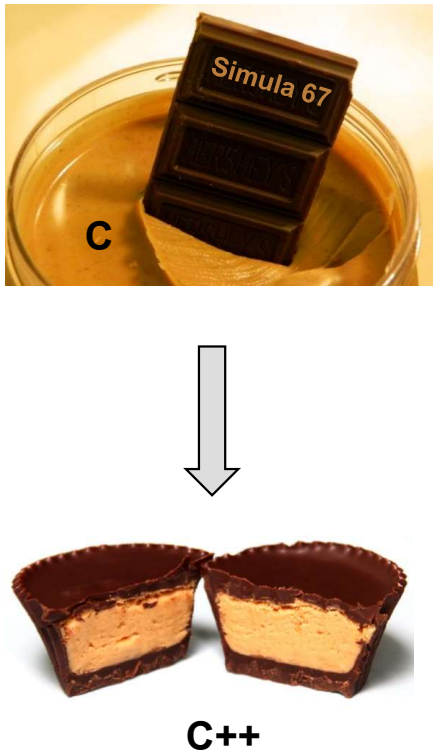
CS 1002 Programming Fundamentals
Lecture 07 12-Sept 2022

National University
Of Computer and Emerging Sciences

Very brief history of C++

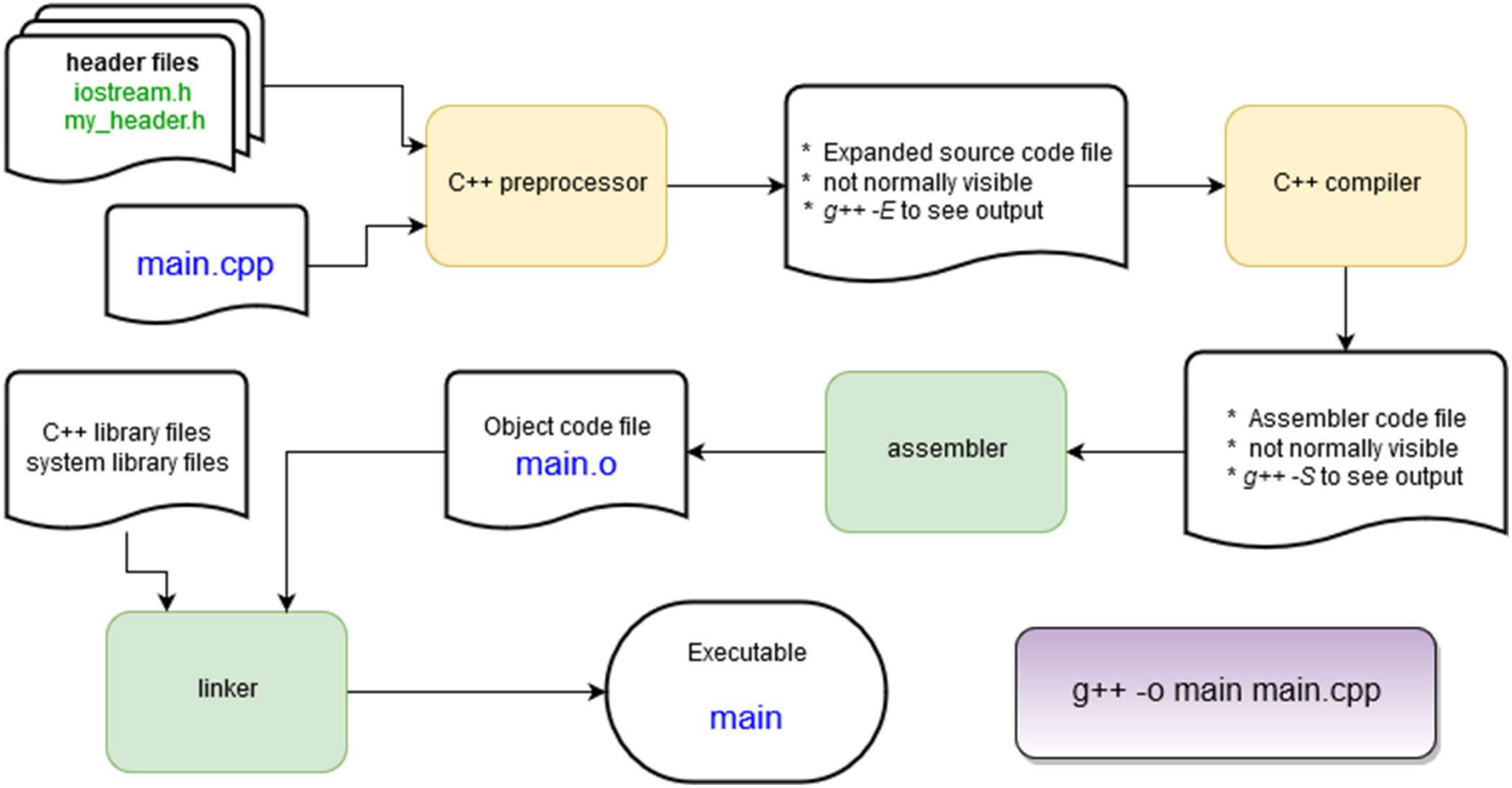




For details more check out A History of C++: 1979−1991

C++ Compilation Process

Behind the Scenes: The Compilation Process



Hello, World! explained

main.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       cout << "Hello world!" << endl;
8       return 0;
9   }
10
```

The *main* routine – the start of **every** C++ program! It returns an integer value to the operating system and (in this case) takes no arguments: main()

The **return** statement returns an integer value to the operating system after completion. 0 means “no error”. C++ programs **must** return an integer value.

Hello, World! explained

loads a *header* file containing function and class definitions

Loads a *namespace* called *std*. Namespaces are used to separate sections of code for programmer convenience. To save typing we'll always use this line in this tutorial.

main.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       cout << "Hello world!" << endl;
8       return 0;
9   }
10
```

- *cout* is the *object* that writes to the stdout device, i.e. the console window.
- It is part of the C++ standard library.
- Without the "using namespace std;" line this would have been called as *std::cout*. It is defined in the *iostream* header file.
- << is the C++ *insertion operator*. It is used to pass characters from the right to the object on the left. *endl* is the C++ newline character.

Header Files

- C++ (along with C) uses *header files* as to hold definitions for the compiler to use while compiling.
- A source file (file.cpp) contains the code that is compiled into an object file (file.o).
- The header (file.h) is used to tell the compiler what to expect when it assembles the program in the linking stage from the object files.
- Source files and header files can refer to any number of other header files.

C++ language headers aren't referred to with the .h suffix. <iostream> provides definitions for I/O functions, including the *cout* function.

```
#include <iostream>

using namespace std;

int main()
{
    string hello = "Hello";
    string world = "world!";
    string msg = hello + " " + world ;
    cout <<  msg << endl;
    msg[0] = 'h';
    cout <<  msg << endl;
    return 0;
}
```

Variables and Literals

- Variable: a storage location in memory
  - Has a name and a type of data it can hold
  - Must be defined before it can be used:
  - Example: int item;

```
type variable-name;
```

Meaning: variable <variable-name> will be a variable of type <type>

Where type can be:

```
int        //integer
double        //real number
char       //character
```

Example:

```
int a, b, c;
double x;
int sum;
char my-character;
```

Literals

- <u>Literal</u>: a value that is written into a program's code.

  `"hello, there"` (string literal)

  `12` (integer literal)

Integer Literal in Program

Program 2-9

```
1   // This program has literals and a variable.
2   #include <iostream>
3   using namespace std;
4
5   int main()
6   {
7      int apples;
8
9      apples = 20;
10     cout << "Today we sold " << apples << " bushels of apples.\n";
11     return 0;
12  }
```

20 is an integer literal

Program Output

```
Today we sold 20 bushels of apples.
```

Input statements

```
cin >> variable-name;
```

Meaning: read the value of the variable called <variable-name> from the user

Example:

```
cin >> a;
cin >> b >> c;
cin >> x;
cin >> my-character;
```

Output statements

`cout << variable-name;`

Meaning: print the value of variable <variable-name> to the user

`cout << "any message ";`

Meaning: print the message within quotes to the user

`cout << endl;`

Meaning: print a new line

Example:

```
cout << a;
cout << b << c;
cout << "This is my character: " << my-
character << " he he he"
    << endl;
```

Whitespaces

- Every C++ program contains whitespaces
  - Include blanks, tabs, and newline characters
- Used to separate special symbols, reserved words, and identifiers
- Proper utilization of whitespaces is important
  - Can be used to make the program readable

The \n Escape Sequence & endl Manipulator

- You can also use the \n escape sequence to start a new line of output. This will produce two lines of output:

```
cout << "Programming is\n";
cout << "fun!";
```

Notice that the `\n` is INSIDE the string.

```
cout << "Programming is" << endl;
cout << "fun!";
```

Same next line using endl manipulator

Common Escape Sequence

Escape Sequence	Name	Description
`\n`	Newline	Causes the cursor to go to the next line for subsequent printing.
`\t`	Horizontal tab	Causes the cursor to skip over to the next tab stop.
`\a`	Alarm	Causes the computer to beep.
`\b`	Backspace	Causes the cursor to back up, or move left one position.
`\r`	Return	Causes the cursor to go to the beginning of the current line, not the next line.
`\\`	Backslash	Causes a backslash to be printed.
`\'`	Single quote	Causes a single quotation mark to be printed.
`\"`	Double quote	Causes a double quotation mark to be printed.

Comments

- Comments are for the reader, not the compiler
- Two types:
  - Single line

```
// This is a C++ program. It prints the sentence:
// Welcome to C++ Programming.
```

  - Multiple line

```
/*
   You can include comments that can
   occupy several lines.
*/
```

The Parts of a C++ Program

```
// sample C++ program
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, there!";
    return 0;
}
```

comment

preprocessor directive to include header file

which namespace to use

beginning of function named `main`

beginning of block for `main`

output statement

string literal

send `0` to operating system

end of block for `main`

Special Characters

Character	Name	Meaning
//	Double slash	Beginning of a comment
#	Pound sign	Beginning of preprocessor directive
< >	Open/close brackets	Enclose filename in #include
( )	Open/close parentheses	Used when naming a function
{ }	Open/close brace	Encloses a group of statements
" "	Open/close quotation marks	Encloses string of characters
;	Semicolon	End of a programming statement

Identifiers

- An identifier is a programmer-defined name for some part of a program: variables, functions, etc.
- Consist of letters, digits, and the underscore character (_)
- Must begin with a letter or underscore
- C++ is case sensitive
  - `NUMBER` is not the same as `number`
- Two **predefined identifiers** are `cout` and `cin`
- Unlike reserved words, predefined identifiers may be redefined, but it is not a good idea

C++ Key Words

You cannot use any of the C++ key words as an identifier. These words have reserved meaning.

Table 2-4 The C++ Key Words

and	continue	goto	public	try
and_eq	default	if	register	typedef
asm	delete	inline	reinterpret_cast	typeid
auto	do	int	return	typename
bitand	double	long	short	union
bitor	dynamic_cast	mutable	signed	unsigned
bool	else	namespace	sizeof	using
break	enum	new	static	virtual
case	explicit	not	static_cast	void
catch	export	not_eq	struct	volatile
char	extern	operator	switch	wchar_t
class	false	or	template	while
compl	float	or_eq	this	xor
const	for	private	throw	xor_eq
const_cast	friend	protected	true	

Identifiers (continued)

- The following are legal identifiers in C++:
  - `first`
  - `conversion`
  - `payRate`

TABLE 2-1 Examples of Illegal Identifiers

Illegal Identifier	Description
`employee Salary`	There can be no space between `employee` and `Salary`.
`Hello!`	The exclamation mark cannot be used in an identifier.
`one+two`	The symbol + cannot be used in an identifier.
`2nd`	An identifier cannot begin with a digit.

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
`totalSales`	Yes	
`total_Sales`	Yes	
`total.Sales`	No	Cannot contain .
`4thQtrSales`	No	Cannot begin with digit
`totalSale$`	No	Cannot contain $

Data Types

- Data type: set of values together with a set of operations
- C++ data types fall into three categories:

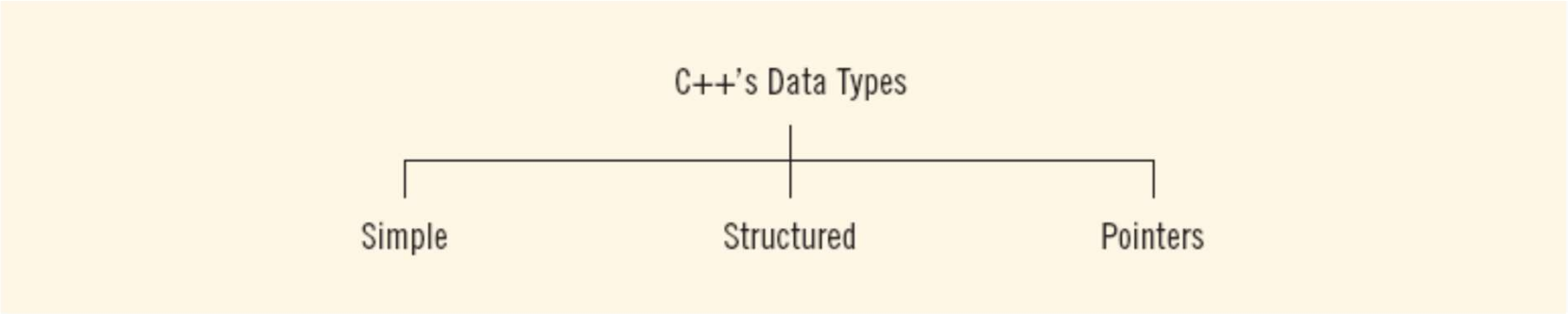


FIGURE 2-1 C++ data types

Simple Data Types

- Three categories of simple data
  - Integral: integers (numbers without a decimal)
  - Floating-point: decimal numbers
  - Enumeration type: user-defined data type

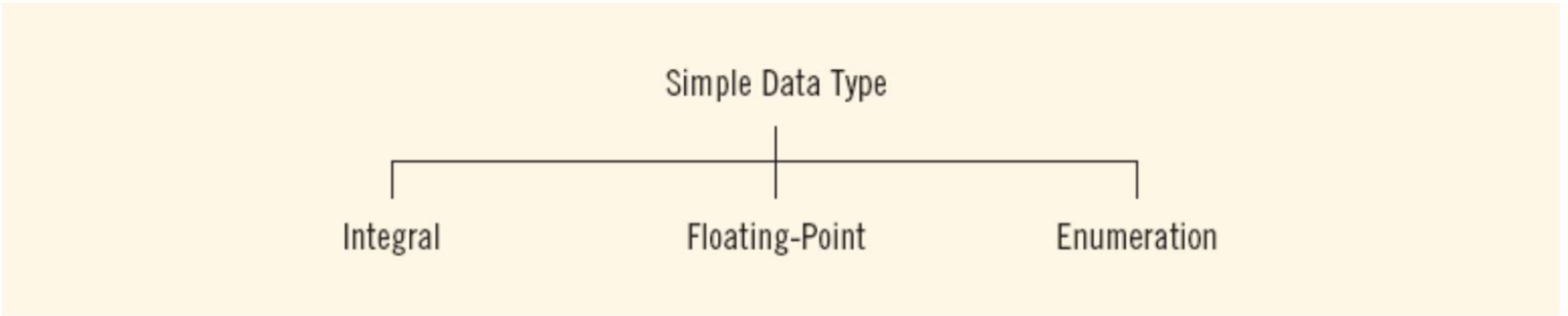


FIGURE 2-2 Simple data types

Simple Data Types (continued)

- Integral data types are further classified into nine categories:

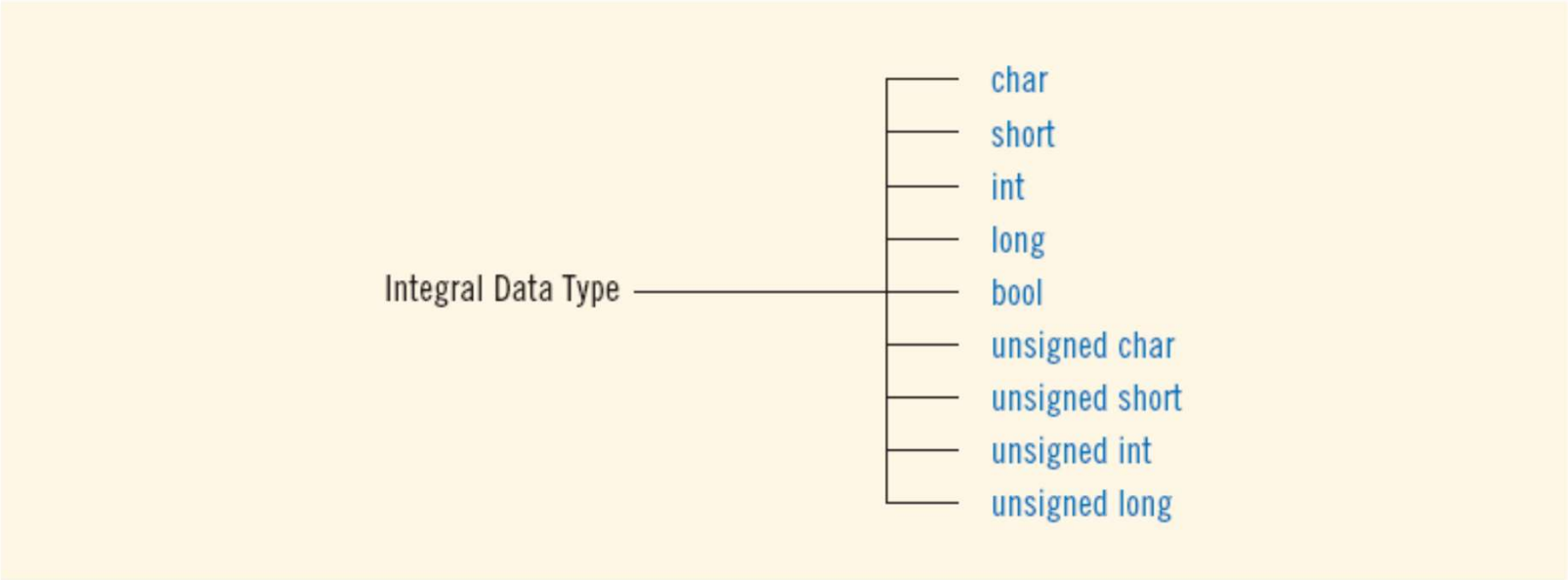


FIGURE 2-3 Integral data types

Simple Data Types (continued)

TABLE 2-2 Values and Memory Allocation for Three Simple Data Types

Data Type	Values	Storage (in bytes)
`int`	-2147483648 to 2147483647	4
`bool`	`true` and `false`	1
`char`	-128 to 127	1

- Different compilers may allow different ranges of values

Variable Type Size Table

Type	Size (in bytes)	Range
char	1	-127 to 127 or 0 to 255
unsigned char	1	0 to 255
int	4	-2147483648 to 2147483647
unsigned int	4	0 to 4294967295
short int	2	-32768 to 32767
unsigned short int	2	0 to 65,535
long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	4	+/- 3.4e +/- 38 (~7 digits)
double	8	+/- 1.7e +/- 308 (~15 digits)

`int` Data Type

- Examples:

  `-6728`

  `0`

  `78`

  `+763`

- Positive integers do not need a `+` sign
- No commas are used within an integer
  - Commas are used for separating items in a list

Integer Literals in Program 2-10

Program 2-10

```
1  // This program has variables of several of the integer types.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int checking;
8     unsigned int miles;
9     long days;
10
11    checking = -20;
12    miles = 4276;
13    days = 189000;
14    cout << "We have made a long journey of " << miles;
15    cout << " miles.\n";
16    cout << "Our checking account balance is " << checking;
17    cout << "\nAbout " << days << " days ago Columbus ";
18    cout << "stood on this spot.\n";
19    return 0;
20 }
```

Integer Literals

`bool` Data Type

- `bool` type
  - Two values: `true` and `false`
  - Manipulate logical (Boolean) expressions
- `true` and `false` are called logical values
- `bool`, `true`, and `false` are reserved words

`char` Data Type

- The smallest integral data type
- Used for <u>characters</u>: letters, digits, and special symbols
- Each character is enclosed in single quotes
  - `'A','a','0','*','+','$','&'`
- A blank space is a character and is written `' '`, with a space left between the single quotes

Boolean Variables in Program 2-17

Program 2-17

```
1   // This program demonstrates boolean variables.
2   #include <iostream>
3   using namespace std;
4
5   int main()
6   {
7      bool boolValue;
8
9      boolValue = true;
10     cout << boolValue << endl;
11     boolValue = false;
12     cout << boolValue << endl;
13     return 0;
14  }
```

Program Output

```
1
0
```

The char Data Type

- Used to hold characters or very small integer values
- Usually 1 byte of memory
- Numeric value of character from the character set is stored in memory:

CODE:

```
char letter;
letter = 'C';
```

MEMORY:

letter

67

- Character literals must be enclosed in single quote marks. Example: `'A'`

Character Strings

- A series of characters in consecutive memory locations:
  - "Hello"
- Stored with the null terminator, \0, at the end:
- Comprised of the characters between the " "

H	e	l	l	o	\0

The C++ string Class

- Special data type supports working with strings
- `#include <string>`
- Can define `string` variables in programs:

```
string firstName, lastName;
```

- Can receive values with assignment operator:

```
firstName = "George";
lastName = "Washington";
```

- Can be displayed via `cout`

```
cout << firstName << " " << lastName;
```

The string class in Program 2-15

Program 2-15

```
// This program demonstrates the string class.
#include <iostream>
#include <string> // Required for the string class.
using namespace std;

int main()
{
   string movieTitle;

   movieTitle = "Wheels of Fury";
   cout << "My favorite movie is " << movieTitle << endl;
   return 0;
}
```

Program Output

```
My favorite movie is Wheels of Fury
```

Floating-Point Data Types

- The floating-point data types are:
  `float`
  `double`
  `long double`

- They can hold real numbers such as:
  `12.45` `-3.8`

- Stored in a form similar to scientific notation

- All floating-point numbers are signed

Decimal Notation	Scientific Notation	E Notation
247.91	2.4791×10^2	`2.4791E2`
0.00072	7.2×10^{-4}	`7.2E-4`
2,900,000	2.9×10^6	`2.9E6`

Floating-Point Data Types

- C++ uses scientific notation to represent real numbers (floating-point notation)

TABLE 2-3 Examples of Real Numbers Printed in C++ Floating-Point Notation

Real Number	C++ Floating-Point Notation
75.924	7.592400E1
0.18	1.800000E-1
0.0000453	4.530000E-5
-1.482	-1.482000E0
7800.0	7.800000E3

Floating-Point Data Types

Table 2-8 Floating Point Data Types on PCs

Data Type	Key Word	Description
Single precision	`float`	4 bytes. Numbers between ±3.4E-38 and ±3.4E38
Double precision	`double`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308
Long double precision	`long double*`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308

`float`: Range: -3.4E+38 to 3.4E+38 (four bytes)

`double`: represents any real number

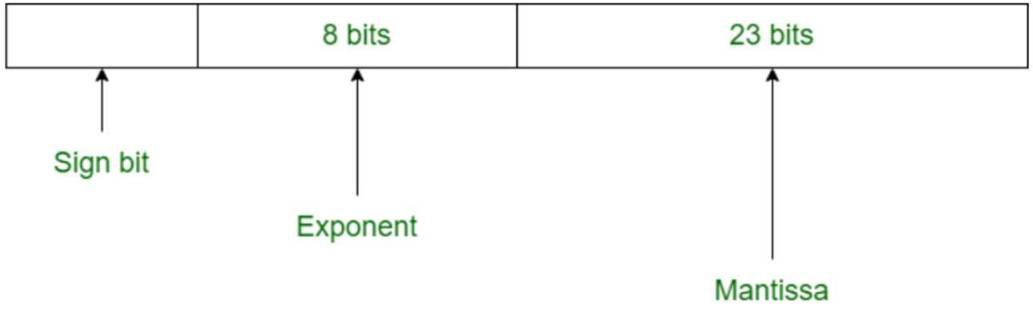
Range: -1.7E+308 to 1.7E+308 (eight bytes)

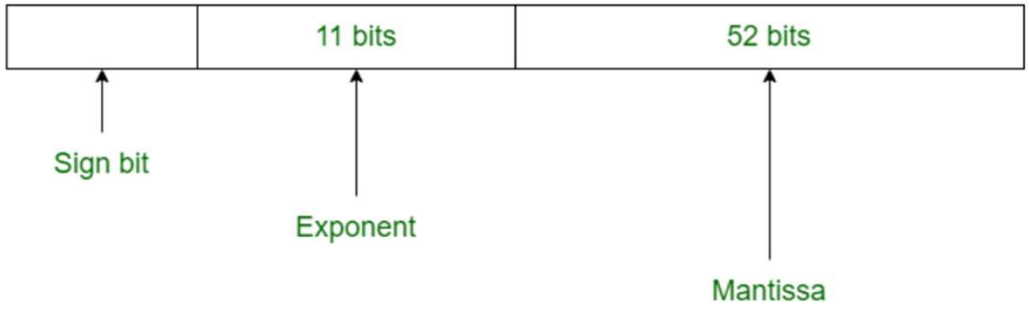
* On most newer compilers, data types `double` and `long double` are same

Floating-Point Data Types (continued)

- Maximum number of significant digits (decimal places) for **float** values is 6 or 7
- Maximum number of significant digits for **double** is 15

- <u>Precision</u>: maximum number of significant digits
  - Float values are called single precision
  - Double values are called double precision





Floating-Point Literals

- Can be represented in
  - Fixed point (decimal) notation:

    `31.4159` `0.0000625`
  - E notation:

    `3.14159E1` `6.25e-5`
- Are `double` by default
- Can be forced to be float (`3.14159f`) or long double (`0.0000625L`)

Variable Assignment and Initialization

- The variable receiving the value must appear on the left side of the = operator.
- This will NOT work:

```
// ERROR!
12 = item;
```

- To initialize a variable means to assign it a value when it is defined:

```
int length = 12;
```

- Can initialize some or all variables:

```
int length = 12, width = 5, area;
```

Scope

- The scope of a variable: the part of the program in which the variable can be accessed
- A variable cannot be used before it is defined

Program 2-20

```
// This program can't find its variable.
#include <iostream>
using namespace std;

int main()
{
   cout << value; // ERROR! value not defined yet!

   int value = 100;
   return 0;
}
```

Arithmetic Operators

- Used for performing numeric calculations
- C++ has unary, binary, and ternary operators:
  - unary (1 operand) -5
  - binary (2 operands) 13 - 7
  - ternary (3 operands) exp1 ? exp2 : exp3

SYMBOL	OPERATION	EXAMPLE	VALUE OF ans
+	addition	`ans = 7 + 3;`	10
-	subtraction	`ans = 7 - 3;`	4
*	multiplication	`ans = 7 * 3;`	21
/	division	`ans = 7 / 3;`	2
%	modulus	`ans = 7 % 3;`	1

Order of Precedence

- All operations inside of () are evaluated first
- *, /, and % are at the same level of precedence and are evaluated next
- + and – have the same level of precedence and are evaluated last
- When operators are on the same level
  - Performed from left to right (associativity)
- `3 * 7 - 6 + 2 * 5 / 4 + 6` means

  `(((3 * 7) - 6) + ((2 * 5) / 4 )) + 6`

A Closer Look at the / Operator

- `/` (division) operator performs integer division if both operands are integers

```
cout << 13 / 5;     // displays 2
cout << 91 / 7;     // displays 13
```

- If either operand is floating point, the result is floating point

```
cout << 13 / 5.0;  // displays 2.6
cout << 91.0 / 7;  // displays 13.0
```


- `/` (division) operator performs integer division if both operands are integers

```
cout << 13 / 5;     // displays 2
cout << 91 / 7;     // displays 13
```

- If either operand is floating point, the result is floating point

```
cout << 13 / 5.0;  // displays 2.6
cout << 91.0 / 7;  // displays 13.0
```

Expressions

- If all operands are integers
  - Expression is called an integral expression
    - Yields an integral result
    - Example: `2 + 3 * 5`
- If all operands are floating-point
  - Expression is called a floating-point expression
    - Yields a floating-point result
    - Example: `12.8 * 17.5 - 34.50`

Mixed Expressions

- Mixed expression:
  - Has operands of different data types
  - Contains integers and floating-point
- Examples of mixed expressions:

```
2 + 3.5
6  /  4 + 3.9
5.4  *  2 - 13.6 + 18  /  2
```

Mixed Expressions (continued)

- Evaluation rules:
  - If operator has same types of operands
    - Evaluated according to the type of the operands
  - If operator has both types of operands
    - Integer is changed to floating-point
    - Operator is evaluated
    - Result is floating-point
  - Entire expression is evaluated according to precedence rules

Named Constants

- Named constant (constant variable): variable whose content cannot be changed during program execution
- Used for representing constant values with descriptive names:

```
const double TAX_RATE = 0.0675;
const int NUM_STATES = 50;
```

- Often named in uppercase letters

Named Constants in Program 2-28

Program 2-28

```
1  // This program calculates the circumference of a circle.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     // Constants
8     const double PI = 3.14159;
9     const double DIAMETER = 10.0;
10
11    // Variable to hold the circumference
12    double circumference;
13
14    // Calculate the circumference.
15    circumference = PI * DIAMETER;
16
17    // Display the circumference.
18    cout << "The circumference is: " << circumference << endl;
19    return 0;
20 }
```

Program Output

```
The circumference is: 31.4159
```

Type Conversion (Casting)

- Implicit type coercion: when value of one type is automatically changed to another type

- Cast operator: provides explicit type conversion

```
static_cast<dataTypeName>(expression)
```

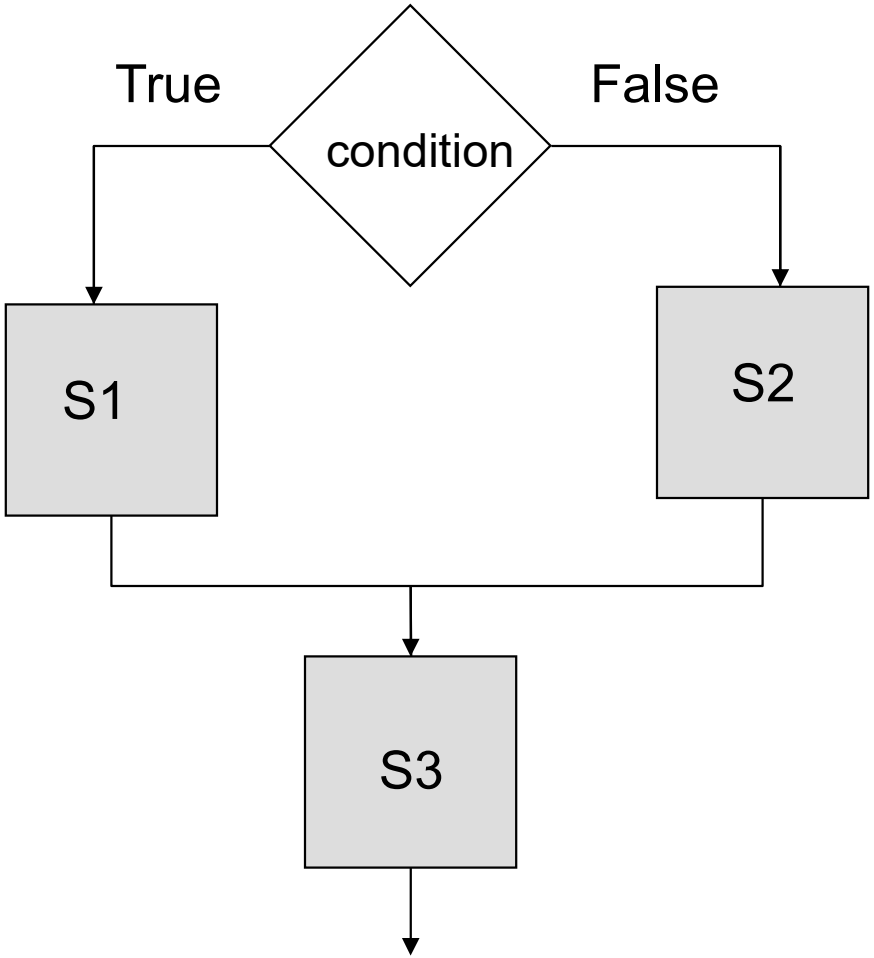
Type Conversion (continued)

EXAMPLE 2-9

Expression	Evaluates to
`static_cast<int>(7.9)`	7
`static_cast<int>(3.3)`	3
`static_cast<double>(25)`	25.0
`static_cast<double>(5+3)`	= `static_cast<double>(8)` = 8.0
`static_cast<double>(15)/2`	= 15.0 / 2 (because `static_cast<double>(15)` = 15.0) = 15.0 / 2.0 = 7.5
`static_cast<double>(15/2)`	= `static_cast<double>(7)` (because 15 / 2 = 7) = 7.0
`static_cast<int>(7.8 +` `static_cast<double>(15)/2)`	= `static_cast<int>(7.8+7.5)` = `static_cast<int>(15.3)` = 15
`static_cast<int>(7.8 +` `static_cast<double>(15/2))`	= `static_cast<int>(7.8 + 7.0)` = `static_cast<int>(14.8)` = 14

If statements

```
if (condition) {
    S1;
}
else {
    S2;
}
S3;
```



Boolean conditions

..are built using

- Comparison operators

==	equal
!=	not equal
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal

- Boolean operators

&&	and
||	or
!	not

If example

```
#include <iostream.h>

void main() {
int a,b,c;
cin >> a >> b >> c;

if (a <=b) {
   cout << "min is " << a << endl;
   }
else {
   cout << " min is " << b << endl;
}
cout << "happy now?" << endl;
}
```

Condition Expression Examples

Assume we declared the following variables:

```
int a = 2, b=5, c=10;
```

Here are some examples of boolean conditions we can use:

- `if (a == b) …`
- `if (a != b) …`
- `if (a <= b+c) …`
- `if(a <= b) && (b <= c) …`
- `if !((a < b) && (b<c)) …`