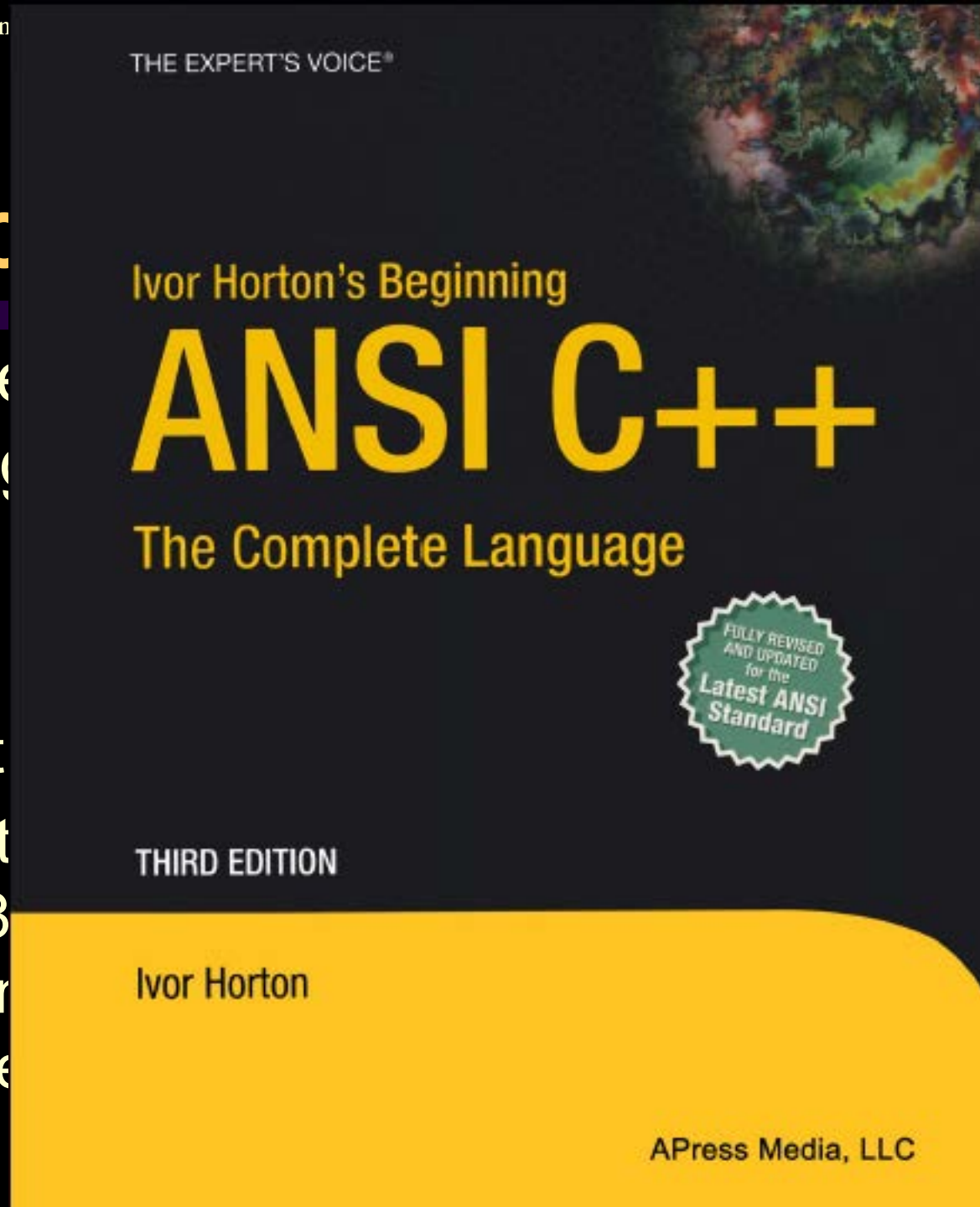


Abc

- Ivor Horton, Be Complete Lang
- ANSI C++
 - The standard
 - American Nat
 - 1998, Internat
 - ISO/IEC 1488
 - INCITS (Intern
 - Information Te



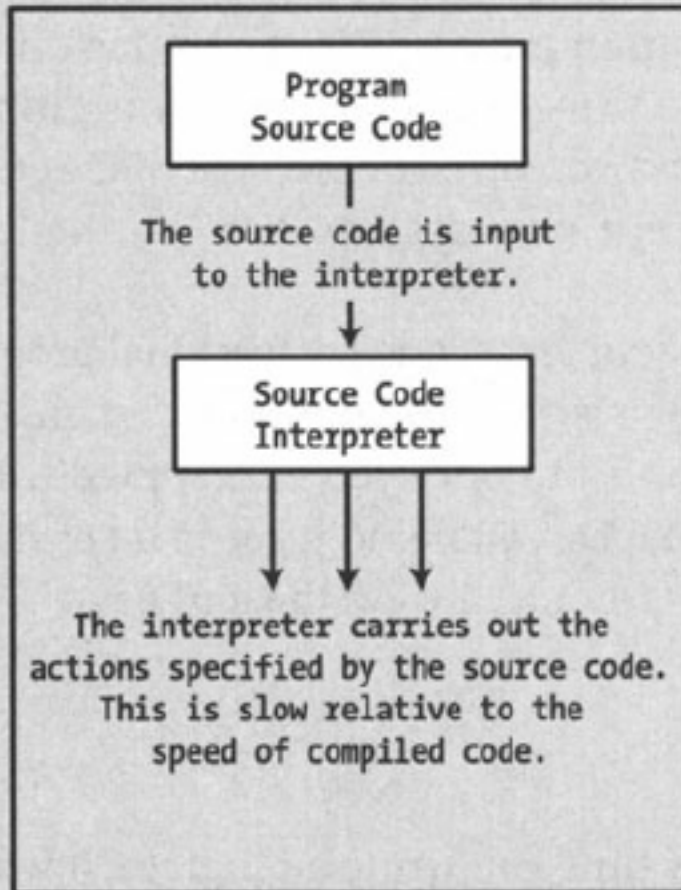
Programming Languages

- High level languages
 - 1950s, **FORTRAN**, LISP, COBOL
 - 1970s, **BASIC**, **C**, PASCAL
 - 1980s, **C++**, Object PACAL/Delphi, Objective-C
 - 1990s, Java, Python
 - 2000s, C#
- Low level language
 - Assembly
- C++ language development environments
 - Borland C++
 - Microsoft Visual C++, Microsoft Turbo C++

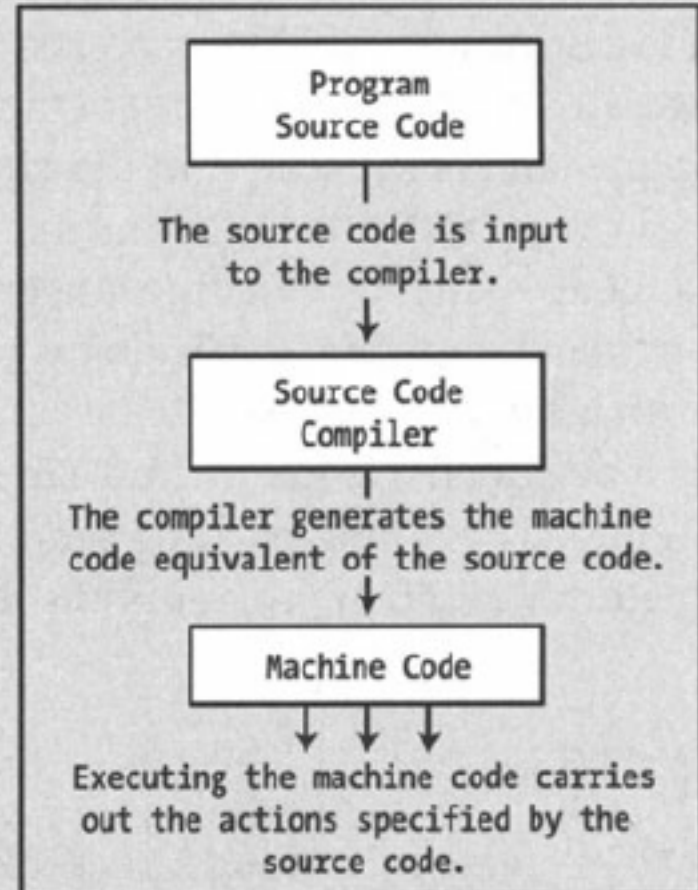
Programming Language rating, 2015,

<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

Interpreter and Compiler



Interpreted Program Execution



Compiled Program Execution

Advantages of C++

- High level language with low level programming (programming down at hardware level)
- Various applications – word processing, scientific computation, operating system, computer games
- High efficient procedural programming inherits from C with powerful object-oriented programming
- Extensive standard library
- Many commercial libraries supporting a wide range of operating system environments and specialized for C++

C++ names

- Names are referred to as identifiers
 - Functions
 - Variables
 - Types
 - Labels
 - Namespaces

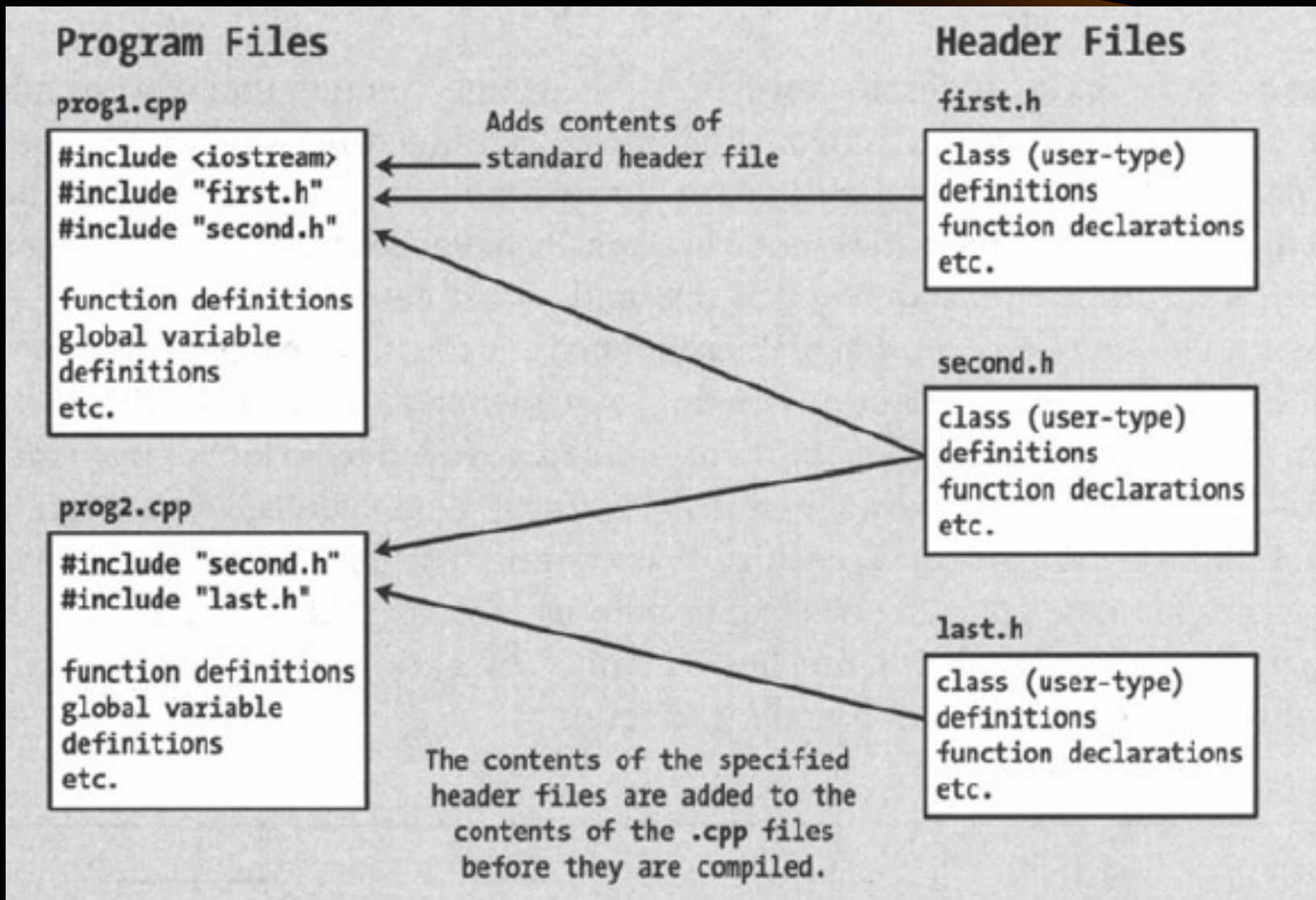
- Legal names

```
value2  Mephistopheles  BettyMay  Earth_Weight  PI
```

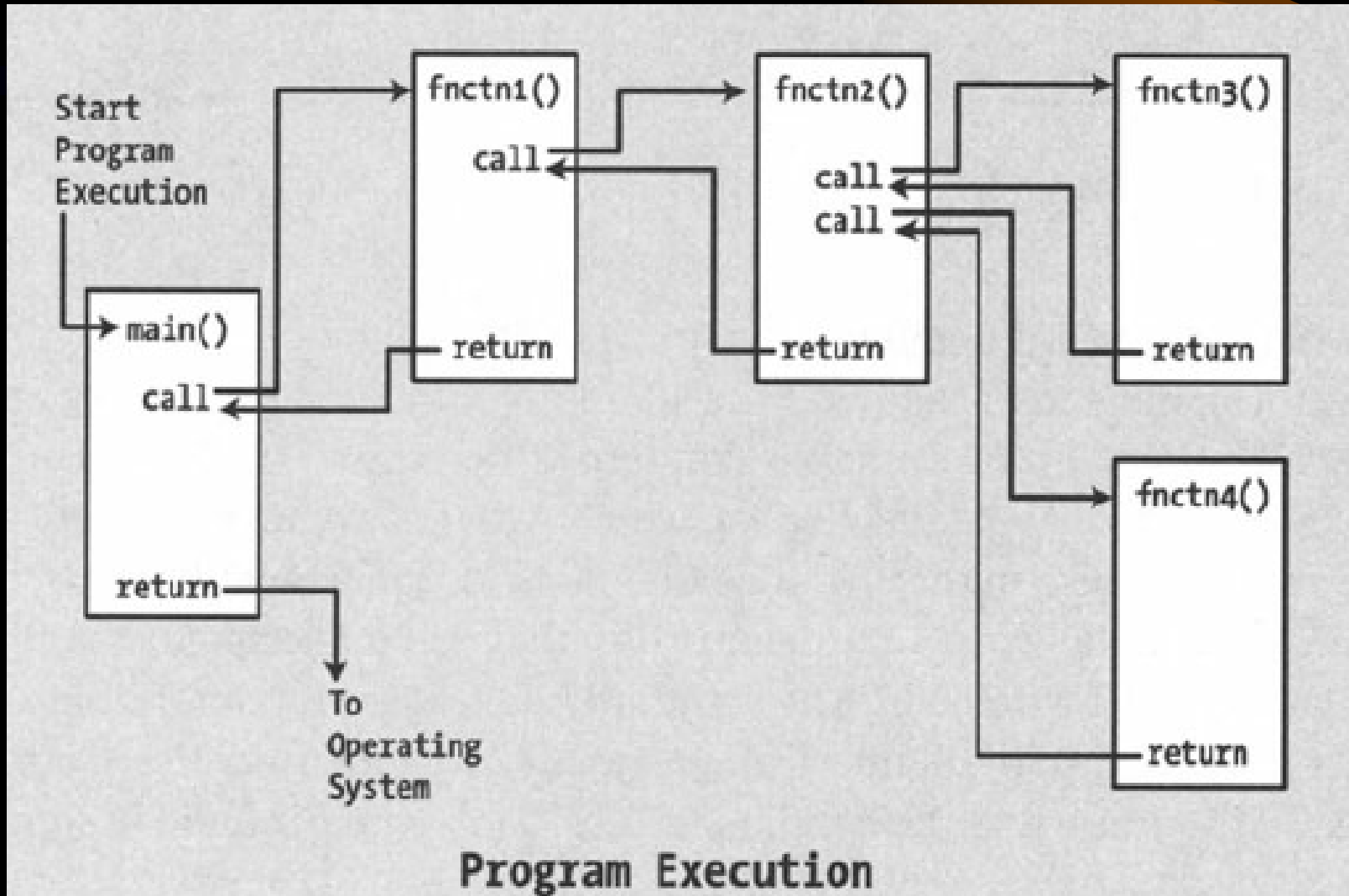
- Illegal names

```
8Ball  Mary-Ann  Betty+May  Earth-Weight  2PI
```

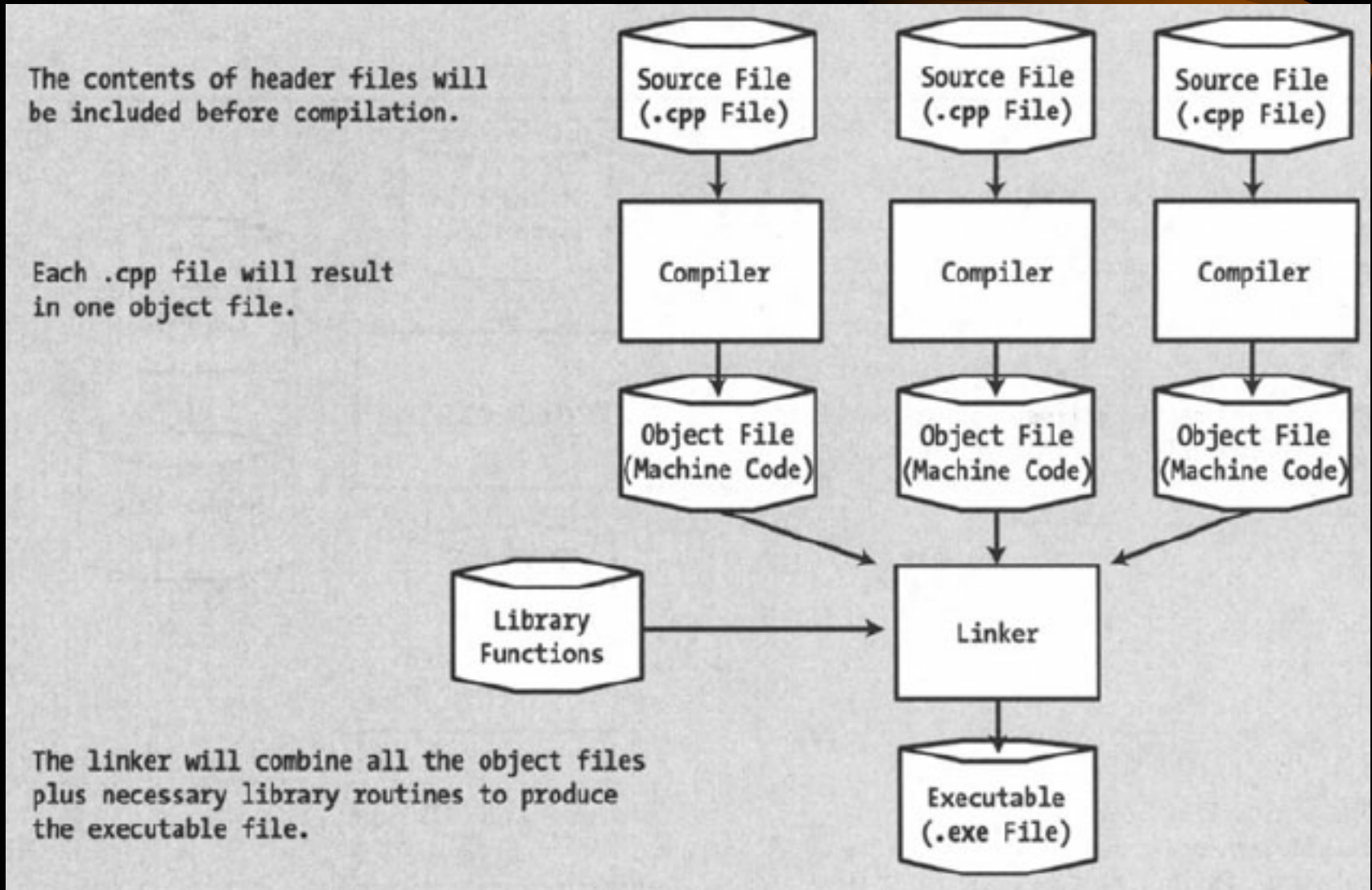
Program Structure



Function Calls and Executions



Compile and Link Processes

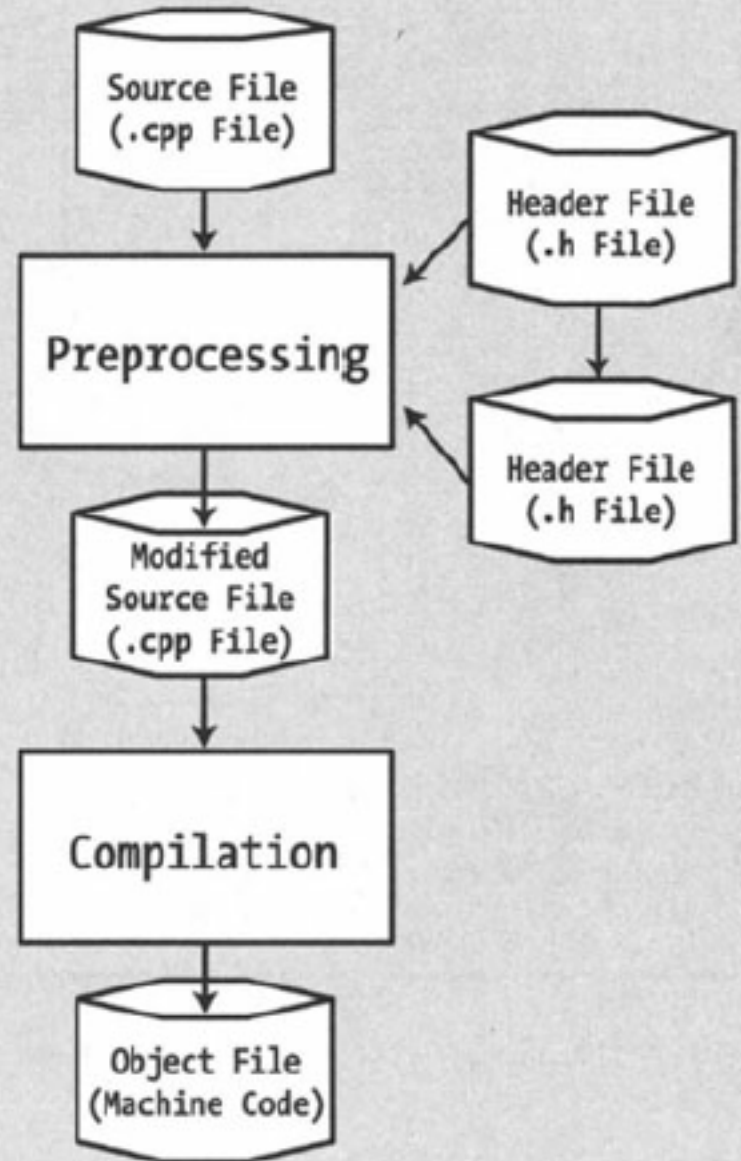


Compilation Process

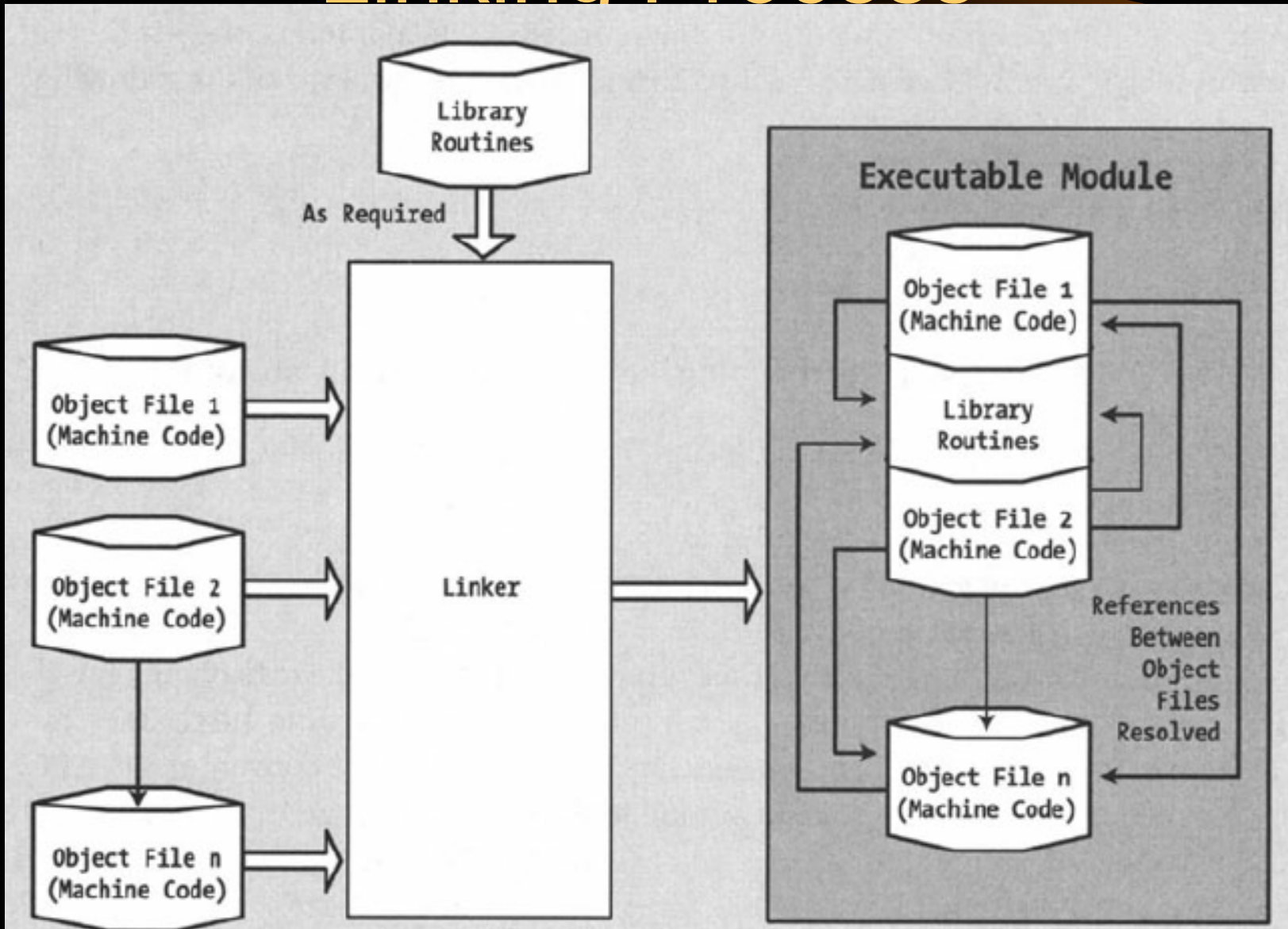
The contents of header files will be included before compilation, as specified by the #include directives.

Other preprocessing directives may modify or conditionally omit some of the code.

The source file processed by the compilation phase is the modified file that contains no preprocessing directives.



Linking Process



C++ Source Characters

- Basic source character set
- ASCII (American Standard Code for Information Interchange)
- Unicode (UCS; Universal Character Set)
 - UCS-2 presents characters as 16-bit codes (65536 different codes)
 - UCS-4 presents characters as 32-bit codes (4 billion different codes)

Escape Sequences

- Control characters

Escape Sequence	Control Character
Escape Sequence	"Problem" Character
\\	Backslash
\'	Single quote
\"	Double quote
\?	Question mark
\t	Form feed
\a	Alert/bell

Simple Programs

- ASCII codes (app. A, p.1009)
 - A hexadecimal number: `\x` or `\X`
 - An octal number: `\`

White space

- When?

```
int fruit;
```

```
fruit = apples + oranges;
```

```
fruit  
=  
apples  
+
```



```
std::cout << std::endl << "\"Least said" << std::endl  
          << "\t\tsoonest mended.\"\\a" << std::endl;
```



```
std::cout << std::endl  
          << "\"Least said"  
          << std::endl  
          << "\t\tsoonest mended.\"\\a"  
          << std::endl;
```


Procedure and Object-oriented Programming

- Procedure programming
 - Also known as **routines**, **subroutines**, **methods**, or **functions**, simply contain a series of computational steps to be carried out
 - C, FORTRAN, PASCAL, COBOL, ...
- Object-oriented programming(OOP)
 - Is a programming paradigm based on the concept of "**object**", which are **data structures** that contain data, in the form of fields, often known as **attributes**; and code, in the form of procedures, often known as **methods**.

The Standard Library



- Standard library header
 - App. C

Namespace std

```
#include <iostream>
using namespace std;
int main()    // int main(void)
{
    cout << "Who is the beginner \?" << endl;
    return 0;
```

```
#include <iostream>
```

```
} #include <iostream>
```

```
using std::cout;
```

```
using std::endl;
```

```
void main() { // void main(void)
```

```
    std::cout << "Who is the beginner \?"
```

```
    << std::endl;
```

```
}
```

```
beginner \?"
```

Variable Types

- Integer variables
 - char (1byte)
 - short (2 bytes)
 - int (2 or 4 bytes)
 - long (4 or 8 bytes)
- Floating variables
 - float (4 bytes)
 - double (8 bytes)
 - long double (8 or 10 bytes)

Integer Domain in VC++

Type	Size (bytes)	Domain
char	1	-128 ~ 127
unsigned char	1	0U ~ 255U
short	2	-32768 ~ 32767
unsigned short	2	0U ~ 65535U
int	4	$-2^{31} \sim 2^{31}-1$
unsigned int	4	$0U \sim 2^{32}-1U$
long	4	$-2^{31} L \sim 2^{31}-1L$
unsigned long	4	$0UL \sim 2^{32}-1UL$

Numeric functions for integers

#include "math.h"

```
int value = -20;  
int result = std::abs(value); // Result is 20
```

or

#include <cmath>

using namespace std;

```
int value = -20;  
int result = abs(value); // Result is 20
```


Table 2-11. <cmath> Numerical Functions

Function	Description
abs(arg)	Returns the absolute value of arg as the same type as arg, where arg can be of any floating-point type. There are versions of the abs() function declared in the <cstdlib> header file for arguments of type int and type long.
fabs(arg)	Returns the absolute value of arg as the same type as the argument. The argument can be int, long, float, double, or long double.
ceil(arg)	Returns a floating-point value of the same type as arg that is the smallest integer greater than or equal to arg, so ceil(2.5) produces the value 3.0. arg can be of any floating-point type.
floor(arg)	Returns a floating-point value of the same type as arg that is the largest integer less than or equal to arg so the value returned by floor(2.5) will be 2.0. arg can be of any floating-point type.
exp(arg)	Returns the value of e^{arg} as the same type as arg. arg can be of any floating-point type.
log(arg)	The log function returns the natural logarithm (to base e) of arg as the same type as arg. arg can be any floating-point type.
log10(arg)	The log10 function returns the logarithm to base 10 of arg as the same type as arg. arg can be any floating-point type.
pow(arg1, arg2)	The pow function returns the value of arg1 raised to the power arg2, which is $\text{arg1}^{\text{arg2}}$. Thus the result of pow(2, 3) will be 8, and the result of pow(1.5, 3) will be 3.375. The arguments can be both of type int or any floating-point type. The second argument, arg2, may also be of type int with arg1 of type int, or long, or any floating-point type. The value returned will be of the same type as arg1.

Table 2-12. `<cmath>` Trigonometric Functions

Function	Description
<code>cos(angle)</code>	Returns the cosine of the angle expressed in radians that is passed as the argument.
<code>sin(angle)</code>	Returns the sine of the angle expressed in radians that is passed as the argument.
<code>tan(angle)</code>	Returns the tangent of the angle expressed in radians that is passed as the argument.
<code>cosh(angle)</code>	Returns the hyperbolic cosine of the angle expressed in radians that is passed as the argument. The hyperbolic cosine of a variable x is given by formula $(e^x + e^{-x})/2$.
<code>sinh(angle)</code>	Returns the hyperbolic sine of the angle expressed in radians that is passed as the argument. The hyperbolic sine of a variable x is given by the formula $(e^x - e^{-x})/2$.
<code>tanh(angle)</code>	Returns the hyperbolic tangent of the angle expressed in radians that is passed as the argument. The hyperbolic tangent of a variable x is given by the hyperbolic sine of x divided by the hyperbolic cosine of x .
<code>acos(arg)</code>	Returns the inverse cosine (arccosine) of <code>arg</code> . The argument must be between -1 and $+1$. The result is in radians and will be from 0 to π .
<code>asin(arg)</code>	Returns the inverse sine (arcsine) of the argument. The argument must be between -1 and $+1$. The result is in radians and will be from $-\pi/2$ to $+\pi/2$.
<code>atan(arg)</code>	Returns the inverse tangent (arctangent) of the argument. The result is in radians and will be from $-\pi/2$ to $+\pi/2$.
<code>atan2(arg1, arg2)</code>	This function requires two arguments of the same floating-point type. The function returns the inverse tangent of <code>arg1/arg2</code> . The result will be in the range from $-\pi$ to $+\pi$ radians and of the same type as the arguments.

Random numbers

```
#include <cstdlib> // standard library
```

```
#include <ctime> // timing function
```

```
std::srand(13); // Set seed for rand to 13
```

```
std::srand((unsigned int)std::time(0)); // random seed
```

- Program 2.5

Floating Numbers

Table 2-7. Floating-Point Number Value

Sign(+/-)	Mantissa	Exponent	Value
-	1.2345	3	-1.2345×10^3 (which is -1234.5)

- In VC++ “-1.2345e003” represent the above number

Floating domain in VC++

Type	Size (bytes)	Decimal precision	Domain
float	4	7	$\approx 10^{-38} \sim 10^{38}$
double	8	15	$\approx 10^{-308} \sim 10^{308}$
long double	8 (10)	15(19)	$\approx 10^{-308} \sim 10^{308}$ ($10^{-4096} \sim 10^{4096}$)

Floating Operation



- The float.h file
- Program 2.7
 - Floating point error
 - Never rely on an exact floating-point representation of a decimal value in your program code
- Program 2.8
 - Floating point output manipulator

Escape sequences

Table 2-13. Escape Sequences for Problem Characters

Character		Escape Sequence
Newline	NL(LF)	\n
Horizontal tab	HT	\t
Vertical tab	VT	\v
Backspace	BS	\b
Carriage return	CR	\r
Form feed	FF	\f
Alert	BEL	\a
Backslash	\	\\
Single quote	'	\'
Double quote	"	\"
Question mark	?	\?

Data Type Conversions

- Static cast
 - `static_cast <object_type> (expression)`
- Constant cast
 - `const_cast <object_type> (expression)`
- Reinterpret cast
 - `reinterpret <object_type> (expression)`
- Dynamic cast
 - `dynamic_cast <object_type> (expression)`

Explicit Casts

- Static cast
 - Old-style casts (C)
 - (object_type) (expression)
 - int a = (int) 3.5; or int a = 3.5;
 - New-style casts (C++)
 - static_cast <object_type> (expression)
 - int a = static_cast <int> 3.5*23;
- Program 3.1

```
long even = 2*static_cast<long>(std::rand());
```



```
long even = 2L* std::rand();
```

```
const int limit = 11;  
int random_value = static_cast<int>(  
    (limit*static_cast<long>(std::rand()))/(RAND_MAX+1L));
```

Mixed expressions



- Mixed expression rule
 - Two floating operation \Rightarrow higher precision
 - Floating and Integer operation \Rightarrow floating

sizeof()



- Count the size of a type or a variable
- Program 3.2

Numeric Limitation

- `numeric_limits <type_name>::min()`
- `numeric_limits <type_name>::max()`
- `numeric_limits <type_name>::digits`
- Program 3.3
- Header file
 - `limits.h`

Boolean Operators

P	Q	P && Q	P Q	!P
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

T: true; F: false

Bitwise operators

Table 3-1. Bitwise Operators

Operator	Description
~	This is the bitwise complement operator . This is a unary operator that will invert the bits in its operand, so 1 becomes 0 and 0 becomes 1.
&	This is the bitwise AND operator , which will AND the corresponding bits in its operands. If the corresponding bits are both 1, then the resulting bit is 1. Otherwise, it's 0.
^	This is the bitwise exclusive OR operator , which will exclusive-OR the corresponding bits in its operands. If the corresponding bits are different (that is, one is 1 and the other is 0), then the resulting bit is 1. If the corresponding bits are the same, the resulting bit is 0.
	This is the bitwise OR operator , which will OR the corresponding bits in its operands. If either of the two corresponding bits is 1, then the result is 1. If both bits are 0, then the result is 0.

Bitwise operations

1: true, 0: false

bit1	bit2	&		^	~bit1
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Bitwise Operators

- Bitwise AND — $\&$
 - $0\&0\Rightarrow 0$ $0\&1\Rightarrow 0$ $1\&1\Rightarrow 1$
- Bitwise OR — $|$
 - $0|0\Rightarrow 0$ $0|1\Rightarrow 1$ $1|1\Rightarrow 1$
- Bitwise XOR — \wedge
 - $0\wedge 0\Rightarrow 0$ $0\wedge 1\Rightarrow 1$ $1\wedge 1\Rightarrow 0$
- Left shift — \ll
 - $(0011\ 1101)\ll 3\Rightarrow (1110\ 1000)$
- Right shift — \gg
 - $(0011\ 1101)\gg 2\Rightarrow (0000\ 1111)$
- Bitwise complement — \sim
 - $\sim(0011\ 1101)\Rightarrow (1100\ 0010)$

Output Manipulators



- Header file
 - `iostream.h`
- Behind `cout <<`
 - `dec`, `hex`, `oct`, `left`, `right`, `fixed`, `scientific`, `showpoint`, `noshowpoint`, `showbase`, `noshowbase`, `showpos`, `noshowpos`, `uppercase`, `nouppercase`, `boolalpha`, `noboolalpha`

Table 3-2. Output Manipulators

Manipulator	Action Performed
dec	Formats integer values as base 10 (decimal). This is the default representation.
hex	Formats integer values as base 16 (hexadecimal).
oct	Formats integer values as base 8 (octal).
left	Left-aligns values in the output field and pads them on the right with the fill character. The default fill character is a space, as you've seen.
right	Right-aligns values in the output field and pads them on the left with the fill character. This is the default alignment.
fixed	Outputs floating-point values in fixed-point notation—that is, without an exponent.
scientific	Outputs floating-point values in scientific notation—that is, as the mantissa plus an exponent. The default mode selects fixed or scientific notation, depending on the value to be displayed.
showpoint	Shows the decimal point and trailing zeros for floating-point values.
noshowpoint	The opposite of the showpoint manipulator. This is the default.
showbase	Prefixes octal output with 0 and hexadecimal output with 0x or 0X.
noshowbase	Shows octal and hexadecimal output without the prefix. This is the default.
showpos	Shows plus signs (+) for positive values.

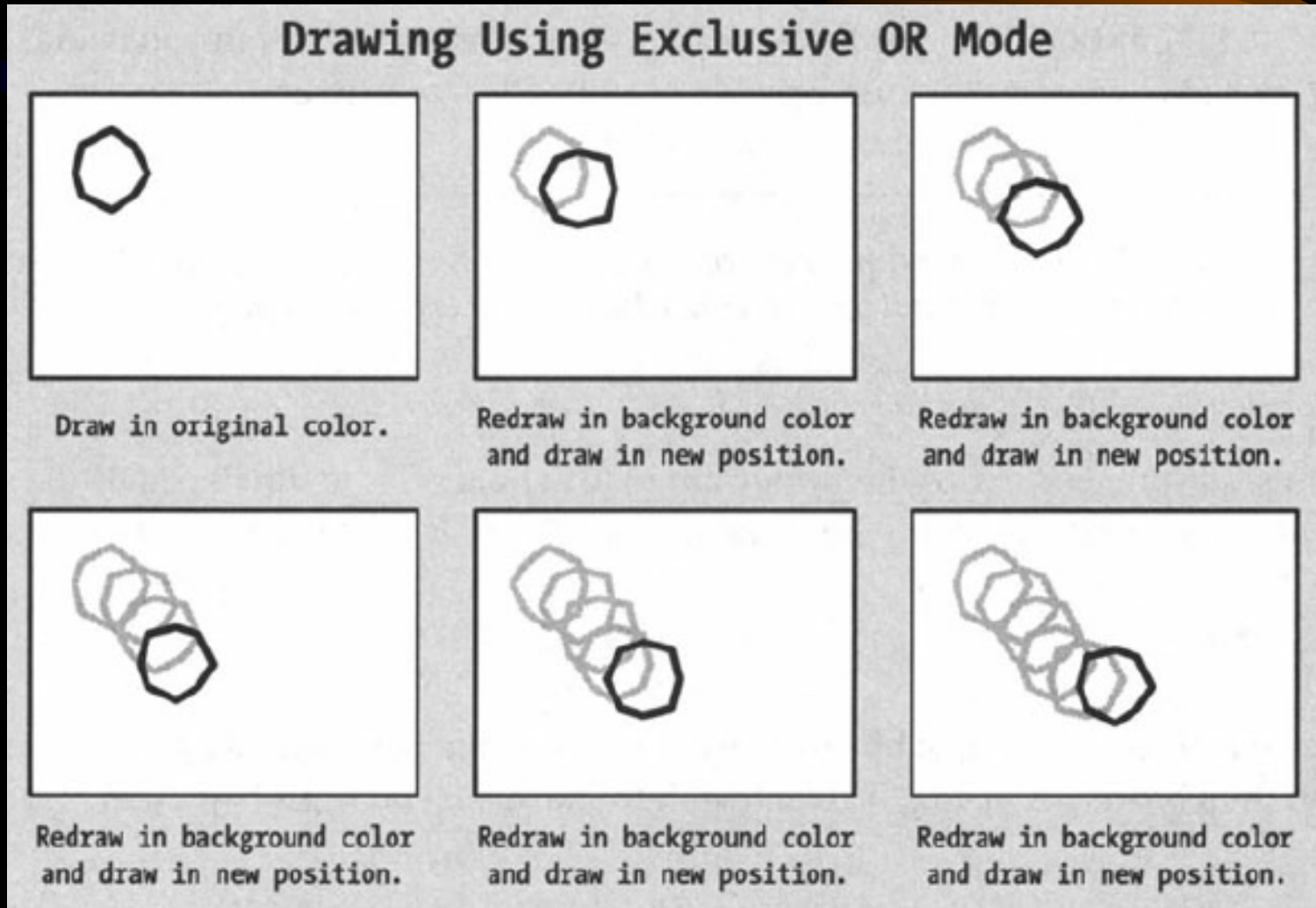
Output Manipulators

- Header file
 - `iomanip.h`
- Behind `cout <<`
 - `setprecision`, `setw`, `setbase`, `setfill`

Table 3-3. Output Manipulators That Require an Argument Value

Manipulator	Action Performed
<code>setfill()</code>	Sets the fill character as specified by the argument. The default fill character is a space.
<code>setw()</code>	Sets the field width as specified by the argument.
<code>setprecision()</code>	Sets the precision for floating-point values as specified by the argument. The precision is the number of decimal digits in the output.

Bitwise exclusive OR operations



- Program 3.4

Lifetime of a Variable

- A variable can have three different kinds of storage duration
 - Automatic storage duration
 - Static storage duration
 - Dynamic storage duration

Variable Scopes

- Automatic variables or local variables
 - Variables declared in block
 - Program 3.6
- Global variables
 - Variables declared outside of blocks
 - :: — scope resolution operator
 - Program 3.7
- Static variables (local access, global exist)
- External variables

Association for the operators

Operator	Association	Operator	Association
Postfix ++	Left	Binary + -	Left
Postfix --	Left	<< >>	Left
Unary +	Right	& ^ ~	Left
Unary -	Right	* / %	Left
Prefix ++	Right	=	Right
Prefix --	Right	static_cast()	Right

Character Testing Functions

- Header file
 - `#include <ctype.h>`
 - `#include <cctype> using namespace std;`
- Character testing functions
 - `isupper()`, `islower()`, `isalpha()`, `isdigit()`,
`isxdigit()`, `isalnum()`, `isspace()`, `iscntrl()`,
`isprint()`, `isgraph()`, `ispunct()`

Logical Operators



- Boolean operators
- Boolean header file
 - `bool.h`
- Return integer
 - false (0), true (non-zero)
- Program 4.4

More coffee and donuts

```
if(coffee == 'y')
    if(donuts == 'y')
        std::cout << "We have coffee and donuts."
                    << std::endl;
    else
        std::cout << "We have coffee, but not donuts."
                    << std::endl;
else if(tea == 'y')
    std::cout << "We have no coffee, but we have tea, and maybe donuts..."
                << std::endl;
else
    std::cout << "No tea or coffee, but maybe donuts..."
                << std::endl;
```

Clearer version

```
if(coffee == 'y') {  
    if(donuts == 'y')  
        std::cout << "We have coffee and donuts."  
                    << std::endl;  
    else  
        std::cout << "We have coffee, but not donuts."  
                    << std::endl;  
}  
else {  
    if(tea == 'y')  
        std::cout << " We have no coffee, but we have tea, and maybe donuts..."  
                    << std::endl;  
    else  
        std::cout << "No tea or coffee, but maybe donuts..."  
                    << std::endl;  
}
```

Logical Operators



- AND (&&), OR(||), NOT(!)
- boolean
 - false (0), true (otherwise)
- Programs 4.7, 4.8, 4.9

switch Statement

```
switch(ticket_number) {  
    case 147:  
        std::cout << "You win first prize!";  
        break;  
    case 387:  
        std::cout << "You win second prize!";  
        break;  
    case 29:  
        std::cout << "You win third prize!";  
        break;  
    default:  
        std::cout << "Sorry, you lose.";  
}
```

Alternative Ways

```
swi switch(tolower(letter)) {  
ca   case 'a': case 'e': case 'i': case 'o': case 'u':  
      cout << endl << "You entered a vowel." << endl;  
      break;  
ca   default:  
      cout << endl << "You entered a consonant." << endl;  
de }  
    cout << endl << "You entered a consonant." << endl;  
}
```


Decision Statement Blocks and Variable Scope

```
if(value > 0) {  
    int savit = value - 1;    // This only exists in this block  
    value += 10;  
}  
else {  
    int savit = value + 1;    // This only exists in this block  
    value -= 10;  
}  
std::cout << savit;          // This will not compile! savit does not exist
```