CSCI 515
Fundamentals Of Programming C/C++

Dr. Eman Hammad
eman.hammad@tamuc.edu

Instructor information

- Dr. Eman Hammad
- Office: RELLIS, ACB1-208 (Bryan/College Station Area)
- Office phone: (979) 317-3432
- Email: eman.hammad@tamuc.edu
- Communication response time <= 24 hours
  - Include the course number/name in the subject field for every email
  - Re-send email if there is no response within 24 hours
  - Response to emails sent on Friday or weekends may be delayed delay. You may not receive any response until Monday.

Course:

- Lectures:
  - Online with a “live” lecture on Tuesdays 3:30-5:00pm via zoom
  - For every lecture,
    - students are expected to do “hands-on” exercises
    - use a problem-driven approach
- Office hours:
  - Wednesdays 10:30am– 12:30pm (zoom) or other times by appointments via emails

Student Learning Outcomes

- to understand the basic elements of a computer program including documentation, data declaration, and procedural operations
- to edit, translate, and execute a computer program
- to write programs that input data from keyboard/file and output to the console/file
- to apply control structures to alter the sequential flow of execution of program statements including selection and iteration structures
- to create user-defined functions, develop programs consisting of multiple functions, and master function parameter passing

- to understand the internal representation of the various data types
- to review the language syntax and learn new syntax you have not previously used in programming applications
- to correctly solve programming problems and learn how to develop algorithms
- to examine the internal representation of two- and three-dimension arrays in C/C++
- to understand dynamic memory allocation, parameter passing, the use of pointers

Ultimate goal:

- Build solid understanding and develop thinking of object-oriented programing and programming skills while building C++ applications so that students can be ready for late courses and their future with certain career objectives

Example 1

- Investment problem
  - You put $10,000 into a bank account that earns 5 percent interest per year.
  - How many years does it take for the account balance to be double the original investment?

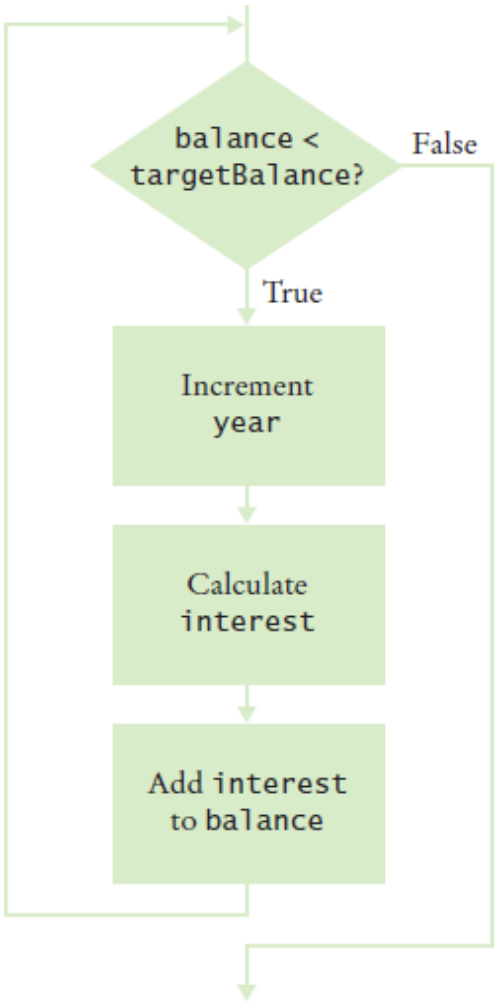
year	interest	balance
0		$10,000

Algorithm

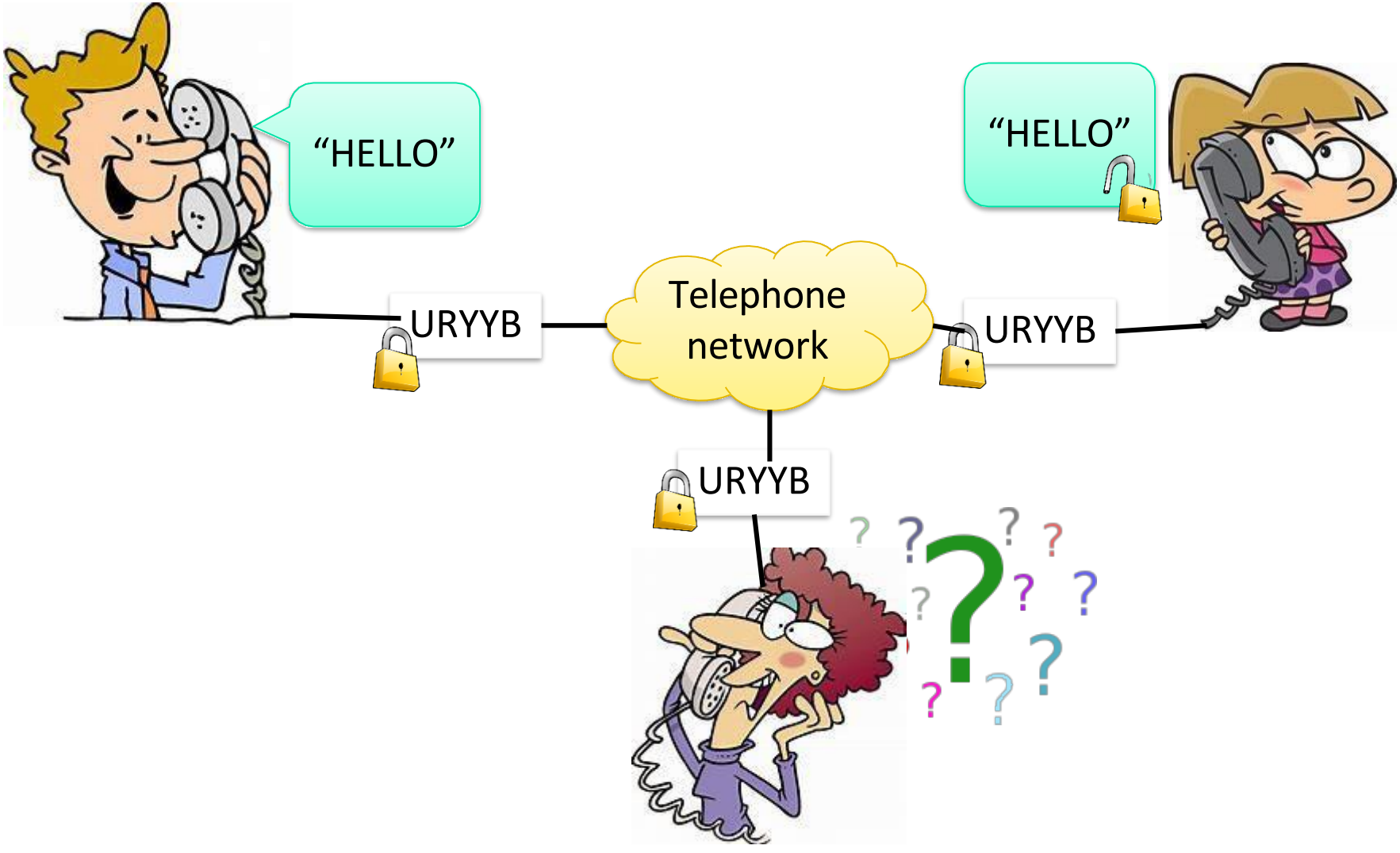
- Start with a year value of 0, a column for the interest, and a balance of $10,000.

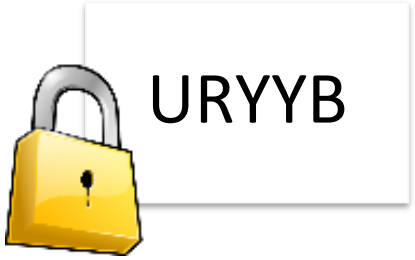
year	interest	balance
0		$10,000

- Repeat the following steps while the balance is less than $20,000.
  - Add 1 to the year value.
  - Compute the interest as balance x 0.05 (i.e., 5 percent interest).
  - Add the interest to the balance.
  - Report the final year value as the answer.



Example 2



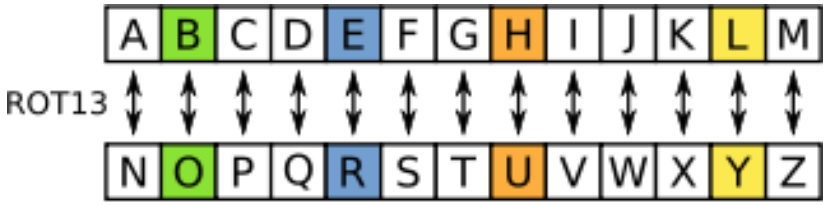


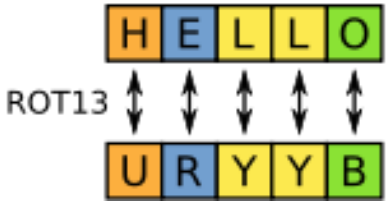
How difficult do you think it would be to break the code?

http://www.cplusplus.com/articles/4iLN8vqX/

Simple Encryption

- Some information on the network may be encrypted with a simple algorithm known as “rot13,” which rotates each character by 13 positions in the alphabet.





http://www.cplusplus.com/articles/4iLN8vqX/

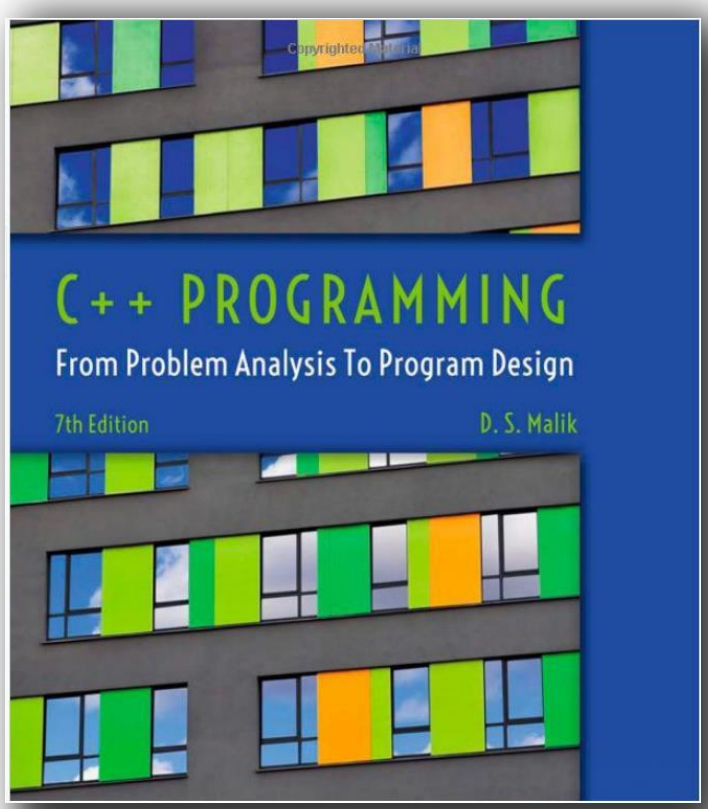
- Lookup table

Input	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
Output	NOPQRSTUVWXYZABCDEFGHIJKLMnopqrstuvwxyzabcdefghijklm

- Inputs:
  - Why did the chicken cross the road?
  - Gb trg gb gur bgure fvqr!

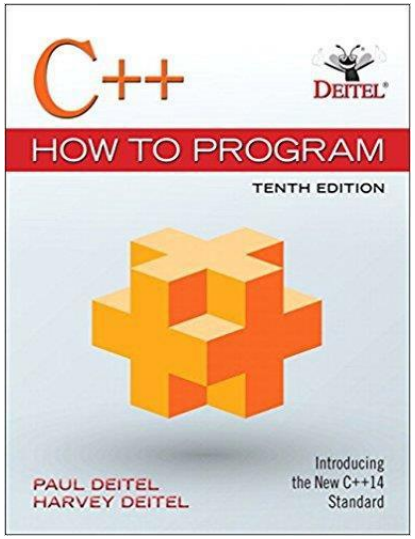
```
#include <cctype>
#include <iostream>
#include <string>

/** http://www.cplusplus.com/articles/4iLN8vqX/
 * \brief     Apply the ROT13 algorithm to a string.
 * \param source Source text to apply the algorithm to.
 * \return    The transformed text is returned.
 */
std::string ROT13(std::string source)
{
        std::string transformed;
        for (size_t i = 0; i < source.size(); ++i) {
                if (isalpha(source[i])) {
                        if ((tolower(source[i]) - 'a') < 14)
                                transformed.append(1, source[i] + 13);
                        else
                                transformed.append(1, source[i] - 13);
                } else {
                        transformed.append(1, source[i]);
                }
        }
        return transformed;
}

int main()
{
        std::string source;
        std::cout << "Enter the source text: " << std::flush;
        std::getline(std::cin, source);
        std::cout << "Result: " << ROT13(source) << std::endl;
        return 0;
}
```

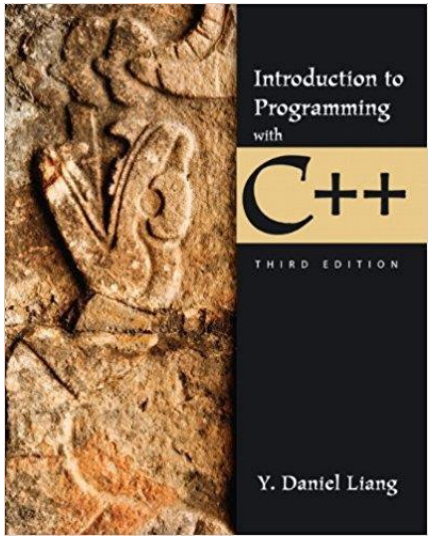
Textbooks

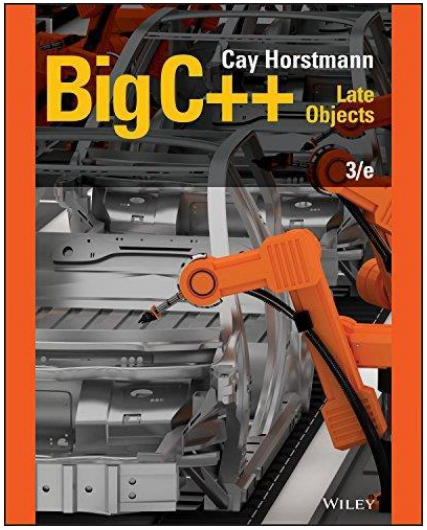


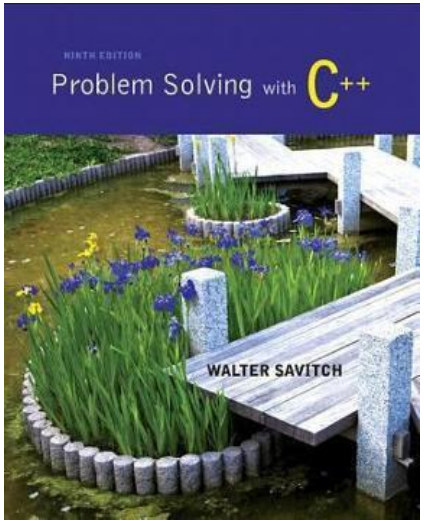
- D . S. Malik, C++ Programming: From Problem Analysis to Program Design, 7th Edition. Cengage Learning, 2015, ISBN 9781285852744
  - Or 8th, or 6th , or 5th edition
- http://www.amazon.com/Programming-Problem-Analysis-Program-Design/dp/1285852745/ref=dp_ob_title_bk

References


C++
DEITEL
HOW TO PROGRAM
TENTH EDITION
PAUL DEITEL
HARVEY DEITEL
Introducing
the New C++14
Standard


Introduction to
Programming
with
C++
THIRD EDITION
Y. Daniel Liang


Big C++
Cay Horstmann
Late
Objects
3/e
WILEY


NINTH EDITION
Problem Solving with C++
WALTER SAVITCH

Topics

- Data types
- Input/output, operators
- Control structures
- Repetition
- Functions
- Enumeration type,
- Typedef
- Namespaces
- Arrays
- Searching and sorting
- Vector type
- Records (struct)
- Pointers
- Classes

- Online resource

  - Tutorialspoint.
    - https://www.tutorialspoint.com/cplusplus/
  - C++ Language – C++ Tutorials
    - http://www.cplusplus.com/doc/tutorial/
  - C++ Tutorial - Learn C++ - Cprogramming.com.
    - https://www.cprogramming.com/tutorial/c++-tutorial.html

In this course,

- Computer Programs
  - Clion- Windows, Linux, and Mac OS
    - https://www.jetbrains.com/clion/
  - Dev-C++/Visual Studio –Windows
    - Dev-C++
      - https://sourceforge.net/projects/orwelldevcpp/
    - Microsoft Visual Studio Express Community 2017 and it is freely available online from Microsoft.
      - https://www.visualstudio.com/vs/visual-studio-express/
  - Eclipse- Windows, Linux, and Mac OS
    - Download from https://eclipse.org/downloads/packages/eclipse-ide- cc-developers/keplersr2
  - Xcode- Mac OS
    - **Note: Visual Studio.net or Dev C++ available in JOUR 101/102**

Assessments

- Assignments/Labs/Quizzes 40%
- Midterm Exam 30%
- Final Exam 30%

Final Grades

A	90% -100%
B	80% - 89%
C	70% - 79%
D	60% - 69%
F	Below 60% (0%-59%)

Prerequisites

- Besides, basic knowledge of computers
- Familiar with any plain text editor (like notepad++)

Logistics

- Computer: desktop or laptop
- Course content: D2L
  - All lecture notes will be available online as well as assignments, labs and other important course information
- Access their email accounts and D2L regularly
- You may be contacted when important matters arise

Rules and policies – review syllabus

- Late submissions policy
- Any indication of copying, cheating or plagiarism on an exam, assignment, or project will be an automatic 0 (zero) for all students involved
- Academic dishonesty will be treated seriously
- Missing classes is at your own risk. Professor will NOT be responsible for any information you might not have due to a missed class
- If you have a disability requiring an accommodation, please contact

Office of Student Disability Resources and Services

- Other TAMUC policies

Syllabus

- All the remain details of the course description, contents, schedule, requirements, grading criteria, assessments, polices, rules are in the syllabus document.

How to succeed in this course?

- Attend/follow lectures and read references if needed be
- Challenge yourself and work on hand-on exercises, assignments and labs on your own
- Attend classes
- Ask questions!!!

Questions or comments?

An Overview of Computers and Programming Languages

Objectives

- Learn about the hardware and software components of a computer system
- Learn about the language of a computer
- Learn about the evolution of programming languages
- Examine high-level programming languages
- Discover what a compiler is and what it does
- Examine a C++ program
- Explore how a C++ program is processed

Anatomy of a computer

- Computers are everywhere
- A computer has two main components
  - Hardware
  - Software

Hardware

- CPU
- Main memory: RAM
  - ROM
- Input/output devices
- Secondary storage
  - Hard drive
- Camera and network interface card

- ROM vs. RAM
  - ROM (Read Only Memory)
    - is non-volatile storage
    - does not require a constant source of power to retain information stored on it. When power is lost or turned off, a ROM chip will keep the information stored on it.
  - RAM (Random Access Memory)
    - is volatile and requires a constant source of power to retain information. When power is lost or turned off, a RAM chip will lose the information stored on it.

Central Processing Unit and Main Memory

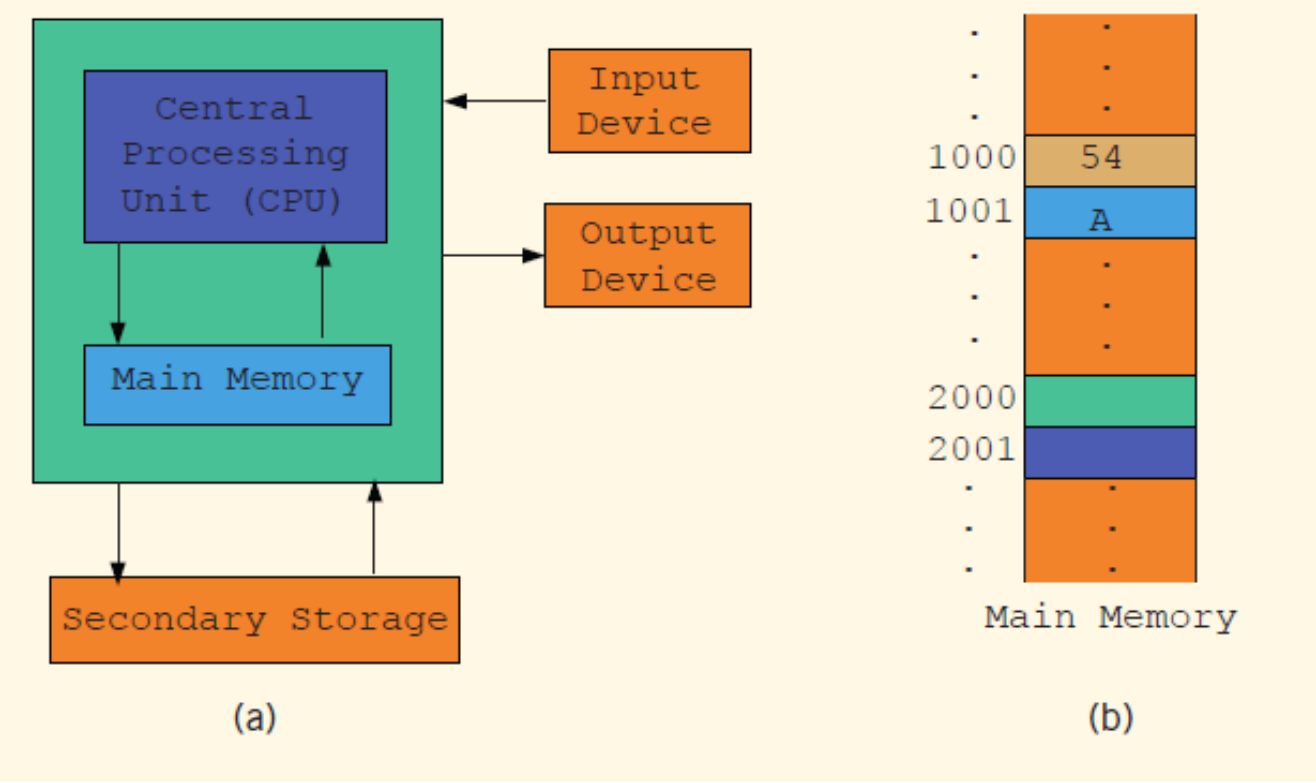


FIGURE 1-1 Hardware components of a computer and main memory

The Language of a Computer

- Analog signals: continuous wave forms
- Digital signals: sequences of 0s and 1s
- Machine language:
  - The language of a computer; a sequence of 0s and 1s
- Binary digit (bit): the digit 0 or 1.
  - A bit is the smallest data item in a computer
- Binary code (binary number): a sequence of 0s and 1s

The Language of a Computer (cont'd.)

- Byte:
  - A sequence of eight bits
- Kilobyte (KB): 2^{10} bytes = 1024 bytes
  - Refer to Table 1-1 for binary unites

TABLE 1-1 Binary Units

Unit	Symbol	Bits/Bytes
Byte		8 bits
Kilobyte	KB	2^{10} bytes = 1024 bytes
Megabyte	MB	1024 KB = 2^{10} KB = 2^{20} bytes = 1,048,576 bytes
Gigabyte	GB	1024 MB = 2^{10} MB = 2^{30} bytes = 1,073,741,824 bytes
Terabyte	TB	1024 GB = 2^{10} GB = 2^{40} bytes = 1,099,511,627,776 bytes
Petabyte	PB	1024 TB = 2^{10} TB = 2^{50} bytes = 1,125,899,906,842,624 bytes
Exabyte	EB	1024 PB = 2^{10} PB = 2^{60} bytes = 1,152,921,504,606,846,976 bytes
Zettabyte	ZB	1024 EB = 2^{10} EB = 2^{70} bytes = 1,180,591,620,717,411,303,424 bytes

- Every letter, number, or speical symbol on your keyboard is encoded as a sequence of bits.
  - 7-bit ASCII (American Standard Code for Information Interchange) –encoding scheme
  - 128 characters numbered 0 through 127
  - A is encoded as 1000001 (66th character)
  - 3 is encoded as 0110011

ASCII (American Standard Code for Information Interchange)

ASCII Table

0	?	1	?	2	?	3	?	4	?	5	?	6	?	7	?
8	?	9	?	10	?	11	?	12	?	13	?	14	?	15	?
16	?	17	?	18	?	19	?	20	?	21	?	22	?	23	?
24	?	25	?	26	?	27	?	28	?	29	?	30	?	31	?
32		33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~		

The Language of a Computer (cont'd.)

- Other encoding schemes
  - EBCDIC
    - Used by IBM
    - 256 characters
  - Unicode
    - 65536 characters
    - Two bytes (i.e., 16 bits) are needed to store a character

The Evolution of Programming Languages

- Early computers were programmed in machine language
- To calculate `wages = rate * hours` in machine language:

```
100100 010001     //Load

100110 010010     //Multiply

100010 010011     //Store
```

Need to remember the code and the address

Shortcoming : difficult and error prone

The Evolution of Programming Languages (cont'd.)

- Assembly language instructions are mnemonic (rely on memory)
- Assembler: translates a program written in assembly language into machine language
- Using assembly language instructions, `wages = rate*`
  - `hours` can be written as:

```
LOAD rate
MULT hour
STOR wages
```

Assembly Language	Machine Language
LOAD	100100
STOR	100010
MULT	100110
ADD	100101
SUB	100011

Shortcoming : remember machine operations

The Evolution of Programming Languages (cont'd.)

- High-level languages include Java, Basic, FORTRAN, COBOL, Pascal, C, Python, C++, and C#
- Compiler: translates a program written in a high-level language into machine language
- The equation `wages = rate • hours` can be written in C++ as:

```
wages = rate * hours;
```

Programming Methodologies

- Two popular approaches to programming design
  - Structured Programming
  - Object Oriented Programming

Structured Programming

- Structured design: (Divide and Conquer)
  - Dividing a problem into smaller subproblems
  - Subproblems are implemented as functions (or further divided)
- Structured programming:
  - Implementing a structured design
- The structured design approach is also called:
  - Top-down (or bottom-up) design
  - Stepwise refinement
  - Modular programming

Object-Oriented Programming

- Object-oriented design (OOD)
  - Identify components called objects
  - Determine how objects interact with each other
- Specify relevant data and possible operations to be performed on that data
- Each object consists of data and operations on that data

Object-Oriented Programming

- An object combines data and operations on the data into a single unit
- A programming language that implements OOD is called an object-oriented programming (OOP) language
- Must learn how to represent data in computer memory, how to manipulate data, and how to implement operations

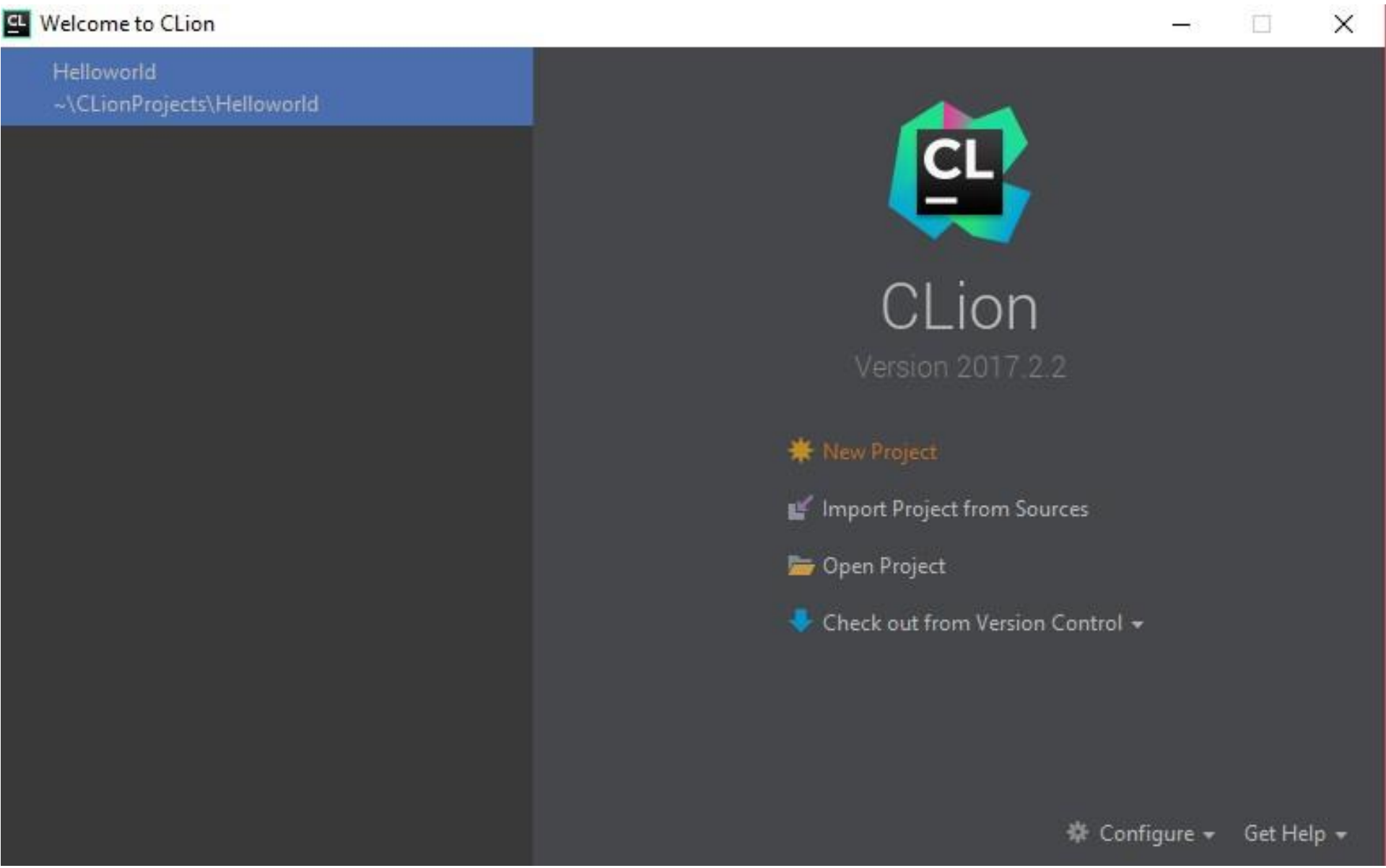
Object-Oriented Programming

- Write algorithms and implement them in a programming language
- Use functions to implement algorithms
- Learn how to combine data and operations on the data into a single unit called an object
- C++ was designed to implement OOD
- OOD is used with structured design

First program

```
#include <iostream>

using namespace std;

int main()

{

     cout << “Hello world." << endl;

    return 0;

}
```

- To execute a C++ program:
  - Use an editor to create a source program in C++
  - Preprocessor directives begin with # and are processed by the preprocessor
  - Use the compiler to:
    - Check if the program obeys the language rules
    - Translate into machine language (object program)


Welcome to CLion
Helloworld
~\CLionProjects\Helloworld
CLion
Version 2017.2.2
New Project
Import Project from Sources
Open Project
Check out from Version Control
Configure
Get Help

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2

(globals)

Project Classes Debug

hw1.cpp

```
1  #include <iostream>
2  using namespace std;
3
4
5  int main()
6  {
7      char name[5] ="John";
8      cout<<name<<endl;
9      string str = "hello";
10     name[6] = 'm';
11
12     int i=0;
13     while(name[i]!='\0')
14     {
15         name[i]='X';
16         i++;
17         cout<<name[i]<<endl;
18     }
19     cout<<name;
20 }
21
```

First program

```
#include <iostream>
using namespace std;
int main()
{
     cout << "Hello world." << endl;
    return 0;
}
```

Preprocessor directives

Sample Run:

```
Hello world.
```

Preprocessor directives

- are commands supplied to the preprocessor program
- are processed before the program goes through the compiler
- All preprocessor commands begin with #
- No semicolon at the end of these commands
- Preprocessor commands

- Syntax :

```
#include <headerFileName>
```

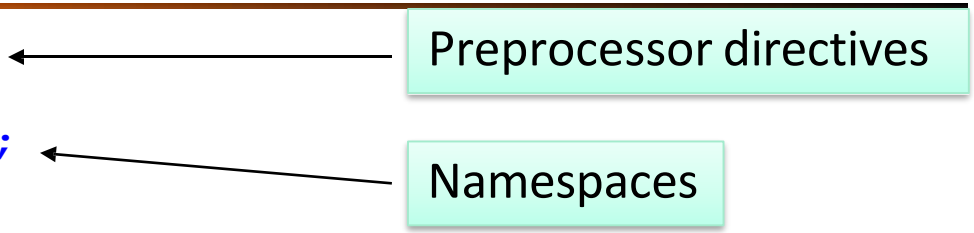
- For example:

```
#include <iostream>
```

  - Causes the preprocessor to include the header file `iostream` in the program

First program

```
#include <iostream>

using namespace std;

int main()

{

     cout << "Hello world." << endl;

    return 0;

}
```



Sample Run:

```
Hello world.
```

Namespaces

- Namespaces are kind of like packages in C++
- Reduces naming conflicts
- Most standard C++ routines and classes are under the `std` namespace
  - Any standard C routines (malloc, printf, etc.) are defined in the global namespace because C doesn't have namespaces

namespace and Using cin and cout in a Program

- cin and cout are declared in the header file iostream, but within std namespace

- To use cin and cout in a program, use the following two statements:

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

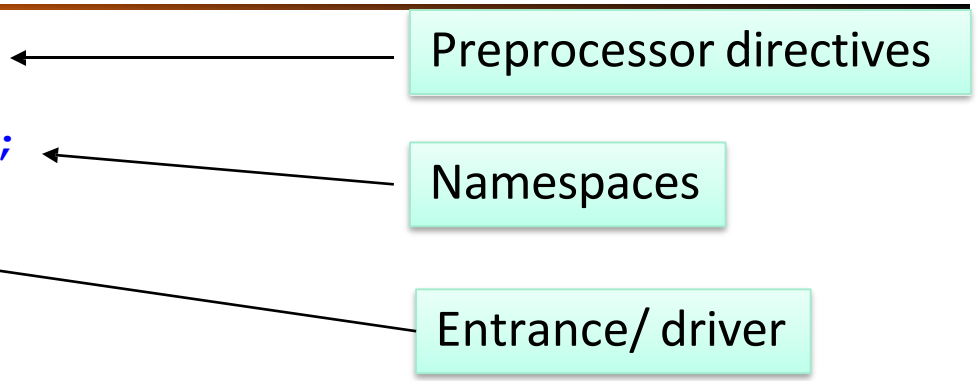
using namespace

```
#include <iostream>

...

std::string question = “How do I prevent RSI?”;

std::cout << question << std::endl;
```

```
using namespace std;

string answer = “Type less.”;

cout << answer << endl;
```

First program

```
#include <iostream>

using namespace std;

int main()

{

     cout << "Hello world." << endl;

    return 0;

}
```



Sample Run:

```
Hello world.
```

```
int main()
{
     //**
}
```

- The function main() is called the entrance/driver.
- This is the first place the computer looks to run code.
- The () indicate that no arguments are passed to the function.
- The “int” means that the program is expecting an integer when we’re done
- {} enclose the code
- Note: Balance parentheses and brackets

- C++ supports two ways of commenting code:

  // line comment

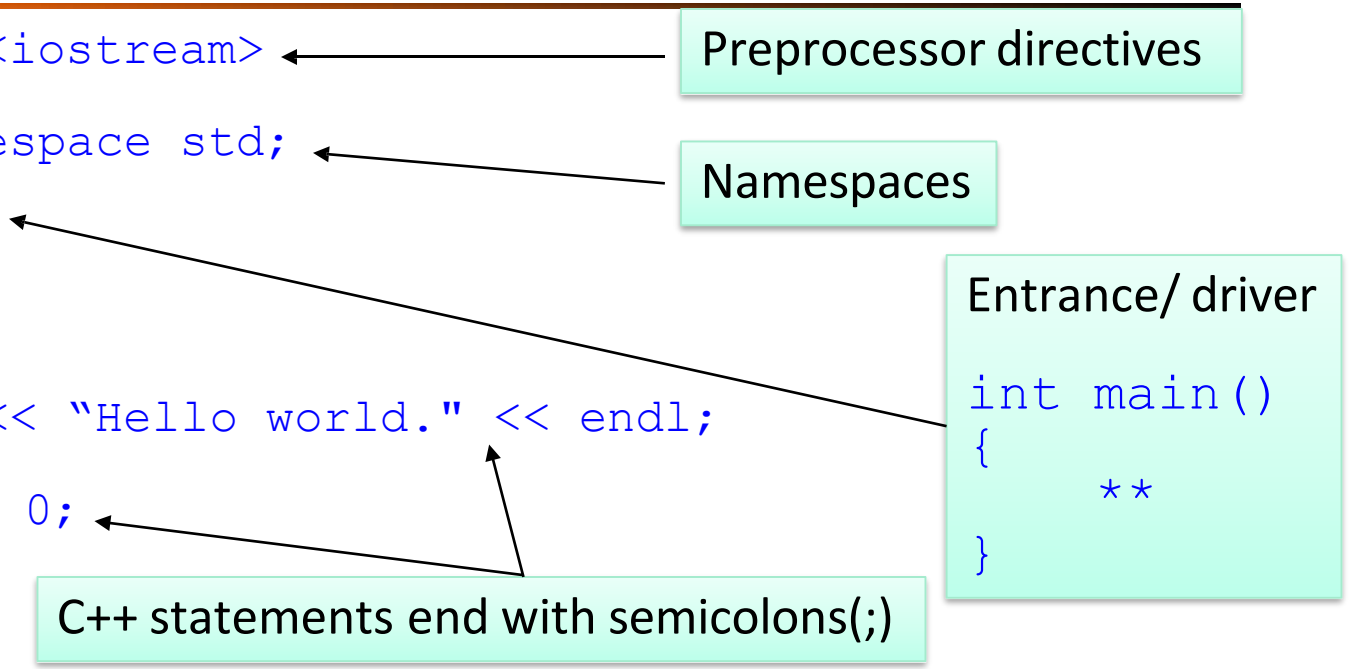
  - discards everything from where the pair of slash signs (//) are found up to the end of that same line.

  /* block comment */

  - discards everything between the /* characters and the first appearance of the */ characters, with the possibility of including multiple lines.

First program

```
#include <iostream>

using namespace std;

int main()

{

     cout << "Hello world." << endl;

    return 0;

}
```



Sample Run:

```
Hello world.
```

- "cout" indicates **the st**andar**d c**haracter **out**put device
  - the computer screen
- The insertion operator (<<) pushes whatever follows to the computer screen
- "endl" means inserts a new-line character ('\n')
- "return" command sends a value to the computer, telling us that the function main is complete

Worksheet

1.) cout << “My precious!\n”;

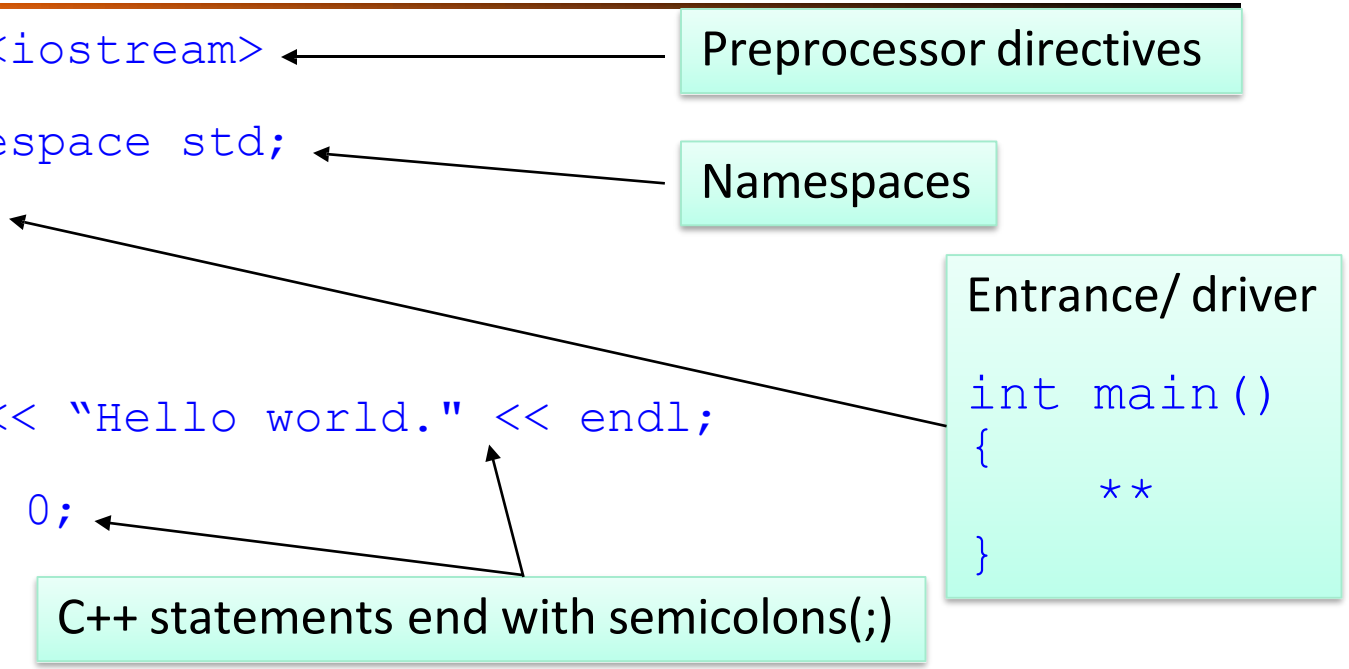
2.) cout << “My \n precious!\n”;

3.) cout << “n\nn”;

4.) cout << “There are 4 hobbits.”;

First program

```
#include <iostream>

using namespace std;

int main()

{

     cout << "Hello world." << endl;

    return 0;

}
```



Sample Run:

```
Hello world.
```

- In the remaining lectures, we will look at input/out, variables, and functions in great detail.

Data type and operator

Dr. Eman Hammad
eman.hammad@tamuc.edu

In the last lecture, you learned

- Explore hardware components of a computer
- Examined a C++ program
- Discovered how a C++ program was processed
- Learn what a compiler is and what it does

In this lecture, you will learn:

- Basic components of a C++ program
  - Functions, special symbols, and identifiers
  - Simple data types
  - Variable declaration
  - Operators
    - Arithmetic operator
    - Increment/ decrement operator (++/--)
  - Basic Input/Output

- Input/Output
  - Learn what a stream is
  - Read data from the standard input device
  - Write data to the standard output device
  - Format output
  - Familiar with file input and output

Problem 1: Modify our first C++ program

```
#include <iostream>

using namespace std;
int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

- Q1: Print multiple lines of text?
  - For example, print “Hello, World!” 5 times on 5 different lines
- Q2: accomplish the above task with a single C++ statement?

- Escape sequences

	Escape Sequence	Description
`\n`	Newline	Cursor moves to the beginning of the next line
`\t`	Tab	Cursor moves to the next tab stop
`\b`	Backspace	Cursor moves one space to the left
`\r`	Return	Cursor moves to the beginning of the current line (not the next line)
`\\`	Backslash	Backslash is printed
`\'`	Single quotation	Single quotation mark is printed
`\"`	Double quotation	Double quotation mark is printed

Comments

- Comments are for the reader, not the compiler
- Two types:
  - Single line (preferred): begin with //

```
// This is a C++ program.
// Welcome to C++ Programming.
```

  - Multiple line: enclosed between /* and */

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

Problem 2

- Q2: write a program to add numbers and display the result.
  - The program should
    - obtain two numbers typed by a user at the keyboard
    - computes their sum and outputs the result.
  - For example:

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

- Basic components of a C++ program
  - Tokens
    - special symbols, world symbols, and identifiers
  - Data types
  - Variable declaration
  - Operators
    - Arithmetic operator
    - Increment/ decrement operator (++/--)

- Basic components of a C++ program
  - **Tokens**
    - **special symbols, world symbols, and identifiers**
  - Data types
  - Variable declaration
  - Operators
    - Arithmetic operator
    - Increment/ decrement operator (++/--)

Token

- Every C++ program has a function called `main`
- Token
  - The smallest individual unit of a program written in any language
  - In C++, tokens are divided into
    - Special symbols
    - Word symbols
    - Identifiers

Symbols

- Special symbols and word symbols
  - Special symbols

+	?
-	,
*	<=
/	!=
.	==
;	>=

  - Word symbols
    - Reserved words, or keywords
      - Include:
        - `int, float, double`
        - `char, void, return`
  - Note
    - **are always lowercase.**
    - **can not be redefined**

Reserved words

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

Symbols (continued)

- Identifiers: the name of something that appears in a program
  - Consist of letters, digits, and the underscore character (_)
  - Must begin with a letter or underscore
  - is case sensitive.
    - `NUMBER` is not the same as `number`
  - Some predefined identifiers are **`cout`** and **`cin`**
  - **Unlike reserved words,**
    - **predefined identifiers may be redefined, but it is not a good idea**

Questions:

- Check legal and illegal identifiers?
  - first
  - conversion
  - payRate
  - Employee Salary
  - Hello!
  - One+two
  - 2nd

Identifiers

- Legal identifiers in C++:
  - `first`
  - `conversion`
  - `payrate`
- Illegal identifiers in C++:

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.

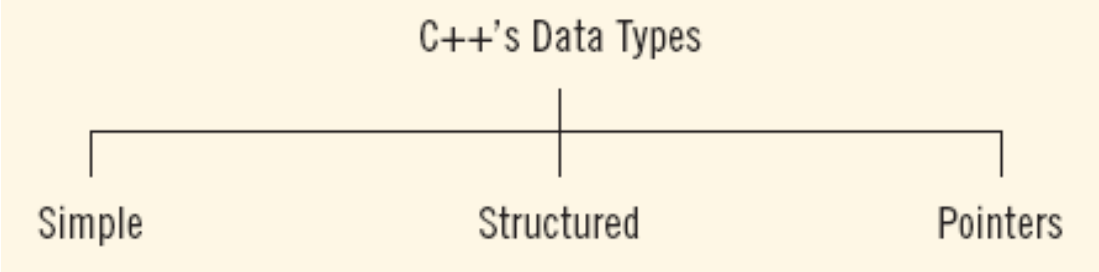
- Basic components of a C++ program
  - Tokens
    - special symbols, and identifiers
  - **Data types**
  - Variable declaration
  - Operators
    - Arithmetic operator
    - Increment/ decrement operator (++/--)

Data Types

- Objective of a C++ program
  - Manipulate data
  - For example,
    - 5+3*2
    - Alphabetize a name list of a class
  - Different types and certain operations
- Thus, we give the definition of data type

Data Types (cont'd.)

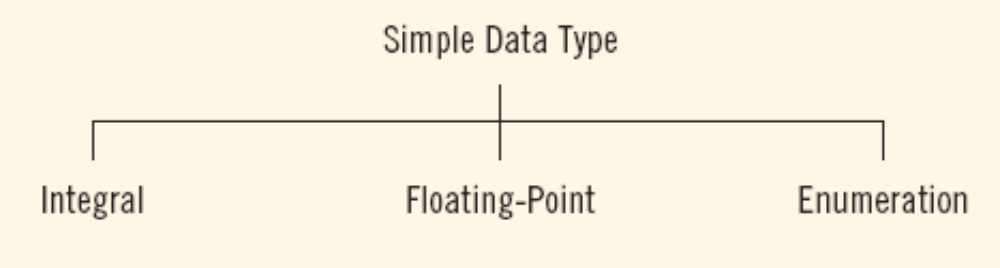
- Data type: a set of values together with a set of operations
- C++ data types fall into three categories:
  - Simple data type (primitive built-in types)
  - Structured data type (struct)
  - Pointers (int*, double*)



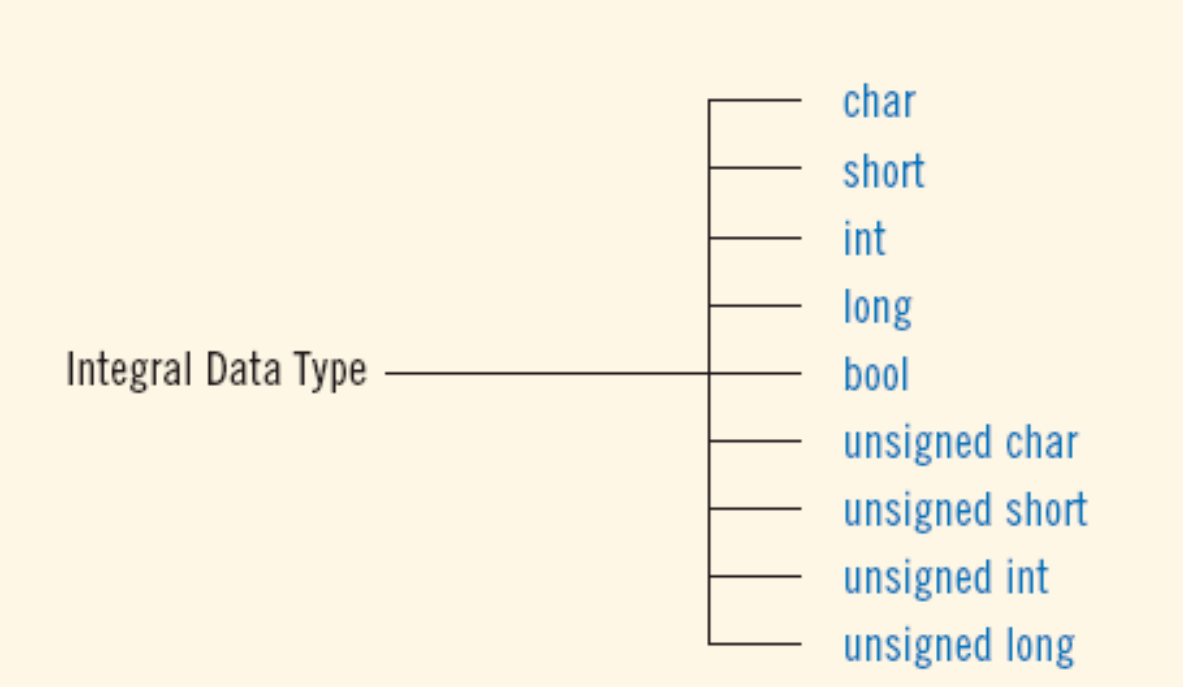
Simple Data Types

- Note: every data type
  - has a different range of possible values
  - Is allocated a different memory size to manipulate the values

- are primitive, built-in or predefined data types and can be used directly by the user to declare variables
  - Example: bool, char, int, float, double, void
- Three categories of simple data
  - Integral: integers (numbers without a decimal)
  - Floating-point: decimal numbers
  - Enumeration type: user-defined data type



- Integral data types are further classified into nine categories:



Simple Data Types

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

- Boolean (bool):
  - is used for storing boolean or logical values, true or false.
- Character (char):
  - is used for storing characters like ‘a’, ‘*’.
  - Each character is enclosed in single quotes
  - Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.
- Integer (int):
  - numbers without a decimal. Integer typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.

- Floating point data type (float/double):
  - is used for storing single/double precision floating point values or decimal values.
  - Floating point variables typically requires 4/8 byte of memory space.
- Precision: maximum number of significant digits
  - Float values are called single precision
    - Maximum number of significant digits (decimal places) for `float` values: 6 or 7
  - Double values are called double precision
    - Maximum number of significant digits for `double`: 15
- Minimum and maximum values of data types are system dependent

Datatype Modifiers

- Datatype modifiers are used with the built-in data types to modify the length of data that a particular data type can hold so that it more precisely fits the needs of various situations.

- Data type modifiers available in C++ are:
  - signed
  - unsigned
  - short
  - long
    - For example: short int, signed short int, long int or long double

Type	size	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	4bytes	-2,147,483,648 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

System dependent

- Sizes might be different in your PC from those shown in the table, depending on the compiler you are using.

How to know

- Check the compiler's documentation
- Run a program

- Example

```
#include <iostream>
using namespace std;
int main()
{
  cout << "Size of char is " << sizeof(char) << endl;
  cout << "Size of int is " << sizeof(int) << endl;
  cout << "Size of float is " << sizeof(float) << endl;
  cout << "Size of short int is " << sizeof(short int) << endl;
  cout << "Size of long int is " << sizeof(long int) << endl;
  cout << "Size of double is " << sizeof(double) << endl;
  cout << "Size of wchar_t is " << sizeof(wchar_t) << endl;

 return 0;
}
```

- sizeof --operator
  - Syntax of using sizeof

```
sizeof(dataType)
```

  - DataType is the desired data type including classes, structures, unions and any other user defined data type.
  - Examples: sizeof(int), sizeof(double)

- Basic components of a C++ program
  - Tokens
    - special symbols, and identifiers
  - Data types
  - Variable declaration
  - Operators
    - Arithmetic operator
    - Increment/ decrement operator (++/--)

Data Types, Variables, and Assignment Statements

- To declare a variable, must specify its data type
- Syntax rule:

```
dataType identifier;
dataType identifier, identifier, identifier,...;
```

- Examples:

```
int counter ;

double interestRate,m;

char grade;
```

- Assignment statement: variable = expression;

```
interestRate = 0.05;
```

Variable Initialization

- 3 ways
  - C-like initialization, constructor initialization, uniform initialization
- Syntax of C-like initialization :

```
dataType identifier = value;
dataType identifier = value, identifier = value,… identifier,…;
```

- Example:

```
/* variable definition and initialization */
int width, height=5, age=32;
char letter='A’;
float area; double d;
/* actual initialization */
width = 10; area = 26.5;

10=20; // invalid
```

- *c-like initialization* inherited from the C language
  - consists of appending an equal sign followed by the value to which the variable is initialized:
    - Example: int x=0;
- constructor initialization introduced by the C++ language
  - encloses the initial value between parentheses (())
    - Syntax: `dataType identifier(value);`
    - Example: int x(0);
- uniform initialization
  - using curly braces ({}) instead of parentheses (this was introduced by the revision of the C++ standard, in 2011)
    - Syntax: `dataType identifier{value};`
    - Example: int x{0};

- All three ways of initializing variables are valid and equivalent in C++

```
// initialization of variables

#include <iostream>
using namespace std;

int main ()
{
  int a=5;
  int b(3);
  int c{2};
  int result;

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

  return 0;
}
```

- Basic components of a C++ program
  - Tokens
    - special symbols, and identifiers
  - Data types
  - Variable declaration
  - Operators
    - Arithmetic operator
    - Increment/ decrement operator (++/--)

`string` Type

- String:
  - Sequence of zero or more characters enclosed in double quotation marks “”
    - "This is a string"
  - Programmer-defined type supplied in ANSI/ISO Standard C++ library
  - Should include the library <string>
  - null (or empty) string: a string with no characters
- Each character has a relative position in the string
  - Position of first character is 0
- Length of a string is number of characters in it
  - Example: length of "`William Jacob`" is 13

Char Vs. String

- A single character inside single quotation marks is char.
  - ‘A’, ‘B’, ‘ ’, ‘5’
  - Be careful! ‘5’ is a char, while 5 is an int.
- A sequence of characters inside double quotation marks is string.
  - “Hello!”
  - “I am happy.”
  - Be careful! ‘5’ is a char, while “5” is a string.
  - Be careful! ‘Hello!’ is illegal.

- What is the output?

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

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

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

```
#include <iostream>
#include <string>

using namespace std;

int main () {

   string str1 = "Hello";
   string str2 = "World";
   string str3;
   int  len ;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total length of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

Arithmetic Operators, Operator Precedence, and Expressions

- C++ arithmetic operators:
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - % modulus (or remainder) operator
- +, -, *, and / can be used with integral and floating-point data types
- Use % only with integral data types

Order of Precedence

- When more than one arithmetic operator is used
  - 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
```

- Online C++ compiler
- www.codepad.org

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

int main()
{
      int test =3*7-6+2*5/4+6;
      cout << " Result:" << test ;
}
```

Expressions

- Integral expression: all operands are integers
  - Yields an integral result
  - Example: `2 + 3 * 5, 15/2`
- Floating-point expression: all operands are floating-points (decimal numbers)
  - Yields a floating-point result
  - Example: `12.8 * 17.5 - 34.50`
- Mixed expression:
  - Has operands of different data types
  - Contains integers and decimal numbers

Mixed Expressions

- Examples of mixed expressions:

```
2 + 3.5

6/4 + 3.9

5.4*2 – 13.6 + 18/2

4 + 5/2.0

4*3 + 7/5 - 25.5
```

Arithmetic Expression	Result
2+5	
13 + 89	
34 - 20	
45 - 90	
2 * 7	
5 / 2	
14 / 7	
34 % 5	
4 % 6	

Arithmetic Expression	Result
5.0+3.5	
3.0+9.4	
16.4-5.2	
4.2*2.5	
5.0/2.0	
36.6/6.0	
34.0%6.0	
34.5/6.0	
36.0%6.0	

Arithmetic Expression	Result
3 * 7 - 6 + 2 * 5 / 4 + 6	
2 + 3.5	
6 / 4 + 3.9	
5.4 * 2 - 13.6 + 18 / 2	
3 / 2 + 5.0	
15.6 / 2 + 5	
4 + 5 / 2.0	
4 * 3 + 7 / 5 - 25.5	
2*7.5/2.5+3	

Arithmetic Expression	Result
2+5	7
13 + 89	102
34 - 20	14
45 - 90	-45
2 * 7	14
5 / 2	2
14 / 7	2
34 % 5	4
4 % 6	4

Arithmetic Expression	Result
5.0+3.5	8.5
3.0+9.4	12.4
16.4-5.2	11.2
4.2*2.5	10.5
5.0/2.0	2.5
36.6/6.0	6.1
34.0%6.0	illegal
34.5/6.0	5.75
36.0%6.0	illegal

Arithmetic Expression	Result
3 * 7 - 6 + 2 * 5 / 4 + 6	23
2 + 3.5	5.5
6 / 4 + 3.9	4.9
5.4 * 2 - 13.6 + 18 / 2	6.2
3 / 2 + 5.0	6.0
15.6 / 2 + 5	12.8
4 + 5 / 2.0	6.5
4 * 3 + 7 / 5 - 25.5	-12.5
2*7.5/2.5+3	9.0

Mixed Expressions (cont'd.)

EXAMPLE 2-8

Mixed Expression	Evaluation	Rule Applied

`3 / 2 + 5.5`	`= 1 + 5.5`
	`= 6.5`
		`3/2 = 1` (integer division; Rule 1(a))
		`(1 + 5.5`
		`= 1.0 + 5.5` (Rule 1(b))
		`= 6.5)`

`15.6 / 2 + 5`	`= 7.8 + 5`
	
	`= 12.8`
		`15.6 / 2`
		`= 15.6 / 2.0` (Rule 1(b))
		`= 7.8`
		`7.8 + 5`
		`= 7.8 + 5.0` (Rule1(b))
		`= 12.8`

`4 + 5 / 2.0`	`= 4 + 2.5`
	
	`= 6.5`
		`5 / 2.0 = 5.0 / 2.0` (Rule1(b))
		`= 2.5`
		`4 + 2.5 = 4.0 + 2.5` (Rule1(b))
		`= 6.5`

`4 * 3 + 7 / 5 - 25.5`	`= 12 + 7 / 5 - 25.5`
	`= 12 + 1 - 25.5`
	`= 13 - 25.5`
	`= -12.5`
		`4 * 3 = 12` (Rule 1(a))
		`7 / 5 = 1` (integer division; Rule 1(a))
		`12 + 1 = 13` (Rule 1(a))
		`13 - 25.5 = 13.0 - 25.5` (Rule 1(b))
		`= -12.5`

Type Conversion (Casting)

- Implicit type conversion: when value of one type is automatically changed to another type
  - Done by the compiler on its own, without any external trigger from the user

```
Examples: double d=3;
          short s=2;
          float division=4.0/3;
          int x='8'+7;
```

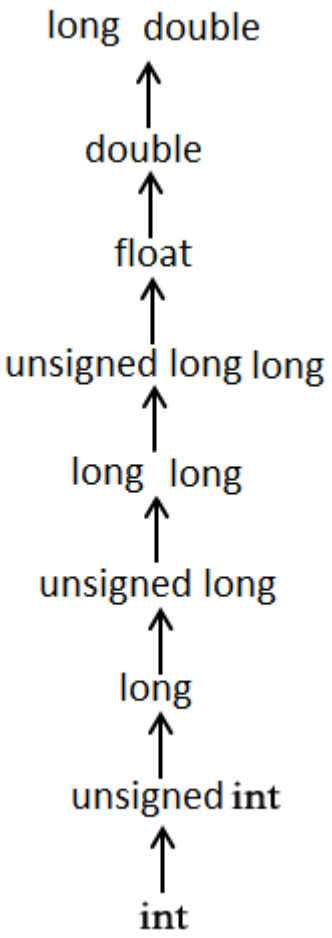
- Cast operator: provides explicit type conversion

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

  - static_cast is the cast operator, which is a reserved word.
  - First, the expression is evaluated.
  - Its value is then converted to a value of the type specified by dataTypeName

Implicit Type Conversion Rules

- The data type of better accuracy is used to store the result. See the chart on the left.
  - 5 + 8.3 is 13.3
- C/C++ is “short-sighted”. It only looks at each operator, not the complete expression.
  - 1.5 + 5 / 10 is 1.5 + 0, so the result is 1.5
  - The program doesn’t look at 1.5 when it is evaluating 5/10, which becomes 0.
- There is no implicit/automatic type conversion for string.



Reserved word

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

Type Conversion (Casting) (cont'd.)

EXAMPLE 2-9

Expression

```
static_cast<int>(7.9)
static_cast<int>(3.3)
static_cast<double>(25)
static_cast<double>(5+3)
static_cast<double>(15)/2


static_cast<double>(15/2)

static_cast<int>(7.8 +
static_cast<double>(15)/2)


static_cast<int>(7.8 +
static_cast<double>(15/2))
```

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

Increment and Decrement Operators

- Increment operator: increase variable by 1
  - Pre-increment: `++variable`
  - Post-increment: `variable++`
- Decrement operator: decrease variable by 1
  - Pre-decrement: `--variable`
  - Post-decrement: `variable–`
- What is the difference between the following?

```
x = 5;
y = ++x;
```

```
x = 5;
y = x++;
```

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;`

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`

Compound operators

- Corresponding to the five arithmetic operators +, -, *, /, and %,
  - C++ provides five **compound operators** +=, -=, *=, /=, and %=.

Operator	*Name*	*Example*	*Equivalent*
+=	Addition assignment	`i += 8`	`i = i + 8`
-=	Subtraction assignment	`i -= 8`	
*=	Multiplication assignment		`i = i * 8`
/=	Division assignment		`i = i / 8`
%=	Remainder assignment		`i = i % 8`

Operator	*Name*	*Example*	*Equivalent*
+=	Addition assignment	`i += 8`	`i = i + 8`
-=	Subtraction assignment	`i -= 8`	`i = i - 8`
*=	Multiplication assignment	`i *= 8`	`i = i * 8`
/=	Division assignment	`i /= 8`	`i = i / 8`
%=	Remainder assignment	`i %= 8`	`i = i % 8`

- x = x / (**4** + **5.5** * **1.5**);

  is equivalent to

  - **The compound operator is performed last after all the other operators in the expression are evaluated**.

Allocating Memory with Constants and Variables

- Named constant: memory location whose content can't change during execution

- Syntax to declare a named constant:

```
const dataType identifier = value;
```

- In C++, `const` is a reserved word

EXAMPLE 2-11

Consider the following C++ statements:

```
const double CONVERSION = 2.54;
const int NO_OF_STUDENTS = 20;
const char BLANK = ' ';
```

A named constant must be initialized when it is declared because from this statement on the compiler will reject any attempt to change the value.

Variable vs constant

- Common:
  - Must be declared with a data type and a legal name
  - Data type will not change
  - Stays in the memory during runtime until the program finishes
- Differences:
  - The value stored by a variable can be changed, but the value stored by a constant cannot be changed.
  - The constant must be assigned with a value in its declaration, but the variable can be assigned later.

Putting Data into Variables

- Ways to place data into a variable:
  - Use C++'s assignment statement
  - Use input (read) statements

Assignment Statement

- The assignment statement takes the form:

```
variable = expression;
```

- Expression is evaluated and its value is assigned to the variable on the left side

- A variable is said to be initialized the first time a value is placed into it

- In C++, = is called the assignment operator

Assignment Statement (cont'd.)

EXAMPLE 2-13

Suppose you have the following variable declarations:

```
int num1, num2;
double sale;
char first;
string str;
```

Now consider the following assignment statements:

```
num1 = 4;
num2 = 4 * 5 - 11;
sale = 0.02 * 1000;
first = 'D';
str = "It is a sunny day.";
```

Input (Read) Statement

- Putting data into variables from the standard input device is accomplished via the use of cin and the operator >>.
  - >> is the stream extraction operator

- **`cin`** is used with **`>>`** to gather input

```
cin >> variable >> variable ...;
```

- This is called an input (read) statement
- For example, if miles is a double variable

```
cin >> miles;
```

  - Causes computer to get a value of type `double` and places it in the variable `miles`

- `>>` distinguishes between character '`2`' and number `2` by the right-side operand of `>>`
  - If type `char` or `int` (or `double`), the `2` is treated as a character or as a number `2`
- Entering a `char` value into an `int` or `double` variable causes serious errors, called input failure

Output

- The syntax of `cout` and `<<` is:

```
cout << expression or manipulator << expression or manipulator...;
```

  - Called an output statement

- The stream insertion operator is `<<`
- `expression` is evaluated
- `value` is printed

Output (cont'd.)

- A manipulator is used to format the output
  - Example: `endl` causes insertion point to move to beginning of next line

EXAMPLE 2-21

Consider the following statements. The output is shown to the right of each statement.

	Statement	Output
1	`cout << 29 / 4 << endl;`	`7`
2	`cout << "Hello there." << endl;`	`Hello there.`
3	`cout << 12 << endl;`	`12`
4	`cout << "4 + 7" << endl;`	`4 + 7`
5	`cout << 4 + 7 << endl;`	`11`
6	`cout << 'A' << endl;`	`A`
7	`cout << "4 + 7 = " << 4 + 7 << endl;`	`4 + 7 = 11`
8	`cout << 2 + 3 * 5 << endl;`	`17`
9	`cout << "Hello \nthere." << endl;`	`Hello` `there.`

Output (cont'd.)

- The new line character is '\n'
  - May appear anywhere in the string

```
cout << "Hello there.";

cout << "My name is James.";
```

Output:

```
Hello there.My name is James.
```

```
cout << "Hello there.\n";

cout << "My name is James.";
```

Output:

```
Hello there.

My name is James.
```

Problem 2

- Q2: write a program to add numbers and display the result.
  - The program should
    - obtain two numbers typed by a user at the keyboard
    - computes their sum and outputs the result.
  - For example:

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


Solution
Analysis
Problem

Literal and Predefined functions

Dr. Eman Hammad
eman.hammad@tamuc.edu

Literals

- A literal is some data that the coder directly typed into the code

- Example:
  - 1
  - 0.5
  - 'A'
  - "Hello“

- A literal has a data type

Data types

- A variable must have a data type.
  - In “double length;”, double is a data type.
- So does a literal.
- Data type: set of values together with a set of operations
- C++ data types fall into three categories:
  - Simple data type
  - Structured data type
  - Pointers

Determine the data type of literal

- All integers are stored as int.
  - 5, 10, -1
- All floating point numbers are stored as double.
  - 0.5, -1.1, 10.0
  - An integer with a decimal point is double.
- If you need to store a number as float, use f.
  - 0.5f is a float.

Determine the data type of literal

- A single character inside single quotation marks is char.
  - ‘A’, ‘B’, ‘ ’, ‘5’
  - Be careful! ‘5’ is a char, while 5 is an int.
- A sequence of characters inside double quotation marks is string.
  - “Hello!”
  - “I am happy.”
  - Be careful! ‘5’ is a char, while “5” is a string.
  - Be careful! ‘Hello!’ is illegal.

Variable vs Literal

- Variable:
  - An identifier with a name. Name cannot change.
  - Must have a data type
  - The data type is specified in its definition
  - Stored in the memory until the program terminates (permanent)
  - Data type will not change during runtime.
  - Data (value) stored can change during runtime.

- Literal:
  - A data that the coder directly typed into the code
  - Must have a data type
  - The data type is determined by its appearance
  - Stored in the memory when it's used. ("Thrown" away after use)
  - "Hard coded". There is nothing for you to change during runtime.

Math (1 of 3)

- In C++, a function is similar to that of a function in algebra
  - It has a name
  - It does some computation
- Predefined Functions
  - Predefined functions are organized into separate libraries
    - I/O functions are in **`iostream`** header
    - Math functions are in **`cmath`** header
  - To use predefined functions, you must include the header file using an **`include`** statement

Math (2 of 3)

- Some of the predefined mathematical functions are:

  pow(x, y), sqrt(x), floor(x)

- See Table in the text for some common predefined functions

Function	Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cmath>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int (double)`	`int (double)`
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than `x`: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle `x`: `cos(0.0) = 1.0`	`double (radians)`	`double`
`exp(x)`	`<cmath>`	Returns e^x, where `e = 2.718`: `exp(1.0) = 2.71828`	`double`	`double`

Math (3 of 3)

Function	Header File	Purpose	Parameter(s) Type	Result
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than `x`: `floor(45.67) = 45.00`	`double`	`double`
`islower(x)`	`<cctype>`	Returns `true` if `x` is a lowercase letter; otherwise it returns `false`; `islower('h')` is `true`	`int`	`int`
`isupper(x)`	`<cctype>`	Returns `true` if `x` is a uppercase letter; otherwise it returns `false`; `isupper('K')` is `true`	`int`	`int`
`pow(x, y)`	`<cmath>`	Returns x^y; If x is negative, `y` must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`sqrt(x)`	`<cmath>`	Returns the nonnegative square root of `x`, `x` must be nonnegative: `sqrt(4.0) = 2.0`	`double`	`double`
`tolower(x)`	`<cctype>`	Returns the lowercase value of `x` if `x` is uppercase; otherwise, returns `x`	`int`	`int`
`toupper(x)`	`<cctype>`	Returns the uppercase value of x if x is lowercase; otherwise, returns `x`	`int`	`int`

`string` Type

- String:
  - Sequence of zero or more characters enclosed in double quotation marks “”
    - "This is a string"
  - Programmer-defined type supplied in ANSI/ISO Standard C++ library
  - Should include the library <string>
  - null (or empty) string: a string with no characters
- Each character has a relative position in the string
  - Position of first character is 0
- Length of a string is number of characters in it
  - Example: length of "`William Jacob`" is 13

- What is the output?

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

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

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

```
#include <iostream>
#include <string>

using namespace std;

int main () {

   string str1 = "Hello";
   string str2 = "World";
   string str3;
   int  len ;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total length of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

string functions

TABLE 7-1 Some `string` functions

Expression	Effect
`strVar.at(index)`	Returns the element at the position specified by `index`.
`strVar[index]`	Returns the element at the position specified by `index`.
`strVar.append(n, ch)`	Appends `n` copies of `ch` to `strVar`, where `ch` is a `char` variable or a `char` constant.
`strVar.append(str)`	Appends `str` to `strVar`.
`strVar.clear()`	Deletes all the characters in `strVar`.
`strVar.compare(str)`	Returns `1` if `strVar > str`; returns `0` if `strVar == str`; returns `-1` if `strVar < str`.
`strVar.empty()`	Returns `true` if `strVar` is empty; otherwise it returns `false`.
`strVar.erase()`	Deletes all the characters in `strVar`.
`strVar.erase(pos, n)`	Deletes `n` characters from `strVar` starting at position `pos`.

TABLE 7-1 Some `string` functions (continued)

Expression	Effect
`strVar.find(str)`	Returns the index of the first occurrence of `str` in `strVar`. If `str` is not found, the special value `string::npos` is returned.
`strVar.find(str, pos);`	Returns the index of the first occurrence at or after `pos` where `str` is found in `strVar`.
`strVar.find_first_of(str, pos)`	Returns the index of the first occurrence of any character of `strVar` in `str`. The search starts at `pos`.
`strVar.find_first_not_of (str, pos)`	Returns the index of the first occurrence of any character of `str` not in `strVar`. The search starts at `pos`.
`strVar.insert(pos, n, ch);`	Inserts `n` occurrences of the character `ch` at index `pos` into `strVar`; `pos` and `n` are of type `string::size_type`; and `ch` is a character.
`strVar.insert(pos, str);`	Inserts all the characters of `str` at index `pos` into `strVar`.
`strVar.length()`	Returns a value of type `string::size_type` giving the number of characters in `strVar`.
`strVar.replace(pos, n, str);`	Starting at index `pos`, replaces the next `n` characters of `strVar` with all the characters of `str`. If `n` > length of `strVar`, then all the characters until the end of `strVar` are replaced.
`strVar.substr(pos, len)`	Returns a string which is a substring of `strVar` starting at `pos`. The length of the substring is at most `len` characters. If `len` is too large, it means "to the end" of the string in `strVar`.
`strVar.size()`	Returns a value of type `string::size_type` giving the number of characters in `strVar`.
`strVar.swap(str1);`	Swaps the contents of `strVar` and `str1`. `str1` is a `string` variable.

- Consider the following statements:

```
string firstName = "Elizabeth";
string name = firstName + " Jones";
string str1 = "It is sunny.";
string str2 = "";
string str3 = "computer science";
string str4 = "C++ programming.";
string str5 = firstName + " is taking " + str4;
string::size_type len;
```

  - Next, we show the effect of **clear**, **empty**, **erase**, **length**, and **size** functions.

Statement	Effect
`str3.clear();`	
`str1.empty();`	
`str2.empty();`	
`str4.erase(11, 4);`	
`cout << firstName.length() << endl;`	
`cout << name.length() << endl;`	
`cout << str1.length() << endl;`	
`cout << str5.size() << endl;`	
`len = name.length();`	

- Consider the following statements:

```
string firstName = "Elizabeth";
string name = firstName + " Jones";
string str1 = "It is sunny.";
string str2 = "";
string str3 = "computer science";
string str4 = "C++ programming.";
string str5 = firstName + " is taking " + str4;
string::size_type len;
```

  - Next, we show the effect of **clear**, **empty**, **erase**, **length**, and **size** functions.

Statement	Effect
`str3.clear();`	`str3 = "";`
`str1.empty();`	Returns `false`
`str2.empty();`	Returns `true`
`str4.erase(11, 4);`	`str4 = "C++ program.";`
`cout << firstName.length() << endl;`	Outputs 9
`cout << name.length() << endl;`	Outputs 15
`cout << str1.length() << endl;`	Outputs 12
`cout << str5.size() << endl;`	Outputs 36
`len = name.length();`	The value of `len` is 15

Operations with strings

```
 //Strings (class) has a set of redefined methods
string str1="Hello";
string str2={'H','e','l','l','o','\0'}; //valid in c++11

string str3;
str3="Hello"; // valid
str3={'H','e','l','l','o','\0'};// valid in c++11

//input
cin>>str3; // stop reading when meeting spaces (whitespaces, tabs, new-
line...) cin.ignore(100,'\n'); // or char ch; cin.get(ch);
//reads until it reaches the end of the current line
getline(cin,str3);

//the size of the string (i.e.,the number of characters)
int n=str1.size();
int m=str1.length();
cout<<n <<" "<<m<<endl;

//output
cout<<str1<<" "<<str2<<" "<<str3<<endl;
for(int i=0;i<6;i++)
    cout<<str3[i];
```

Basic Input / Output

Dr. Eman Hammad
eman.hammad@tamuc.edu

Problem 1

- Q1: Write a program that reads a student name followed by five test scores. The program should output the student name, the five test scores, and the average test score. Output the average test score with two decimal places.
  - The data to be read is stored in a file called test.txt. The output should be stored in a file called result.txt.
  - Input A file containing the student name and the five test scores.
    - A sample input is:

```
Andrew Miller 87.50 89 65.75 37 98.50
```

  - Output The student name, the five test scores, and the average of the five test scores, saved to a file.
    - A sample out is

```
Student name: Andrew Miller
Test scores: 87.50 89.00 65.75 37.00 98.50
Average test score: 75.55
```

- C++ I/O occurs in streams
  - are sequences of bytes.
  - If bytes flow from a device like a keyboard, a disk drive, or a network connection etc. to main memory, this is called input operation
  - if bytes flow from main memory to a device like a display screen, a printer, a disk drive, or a network connection, etc., this is called output operation.

Standard input/output stream

- cin/cout

```
//Example: cin and cout
#include <iostream>
using namespace std;
int main() {
 int g;
 cin>>g;
 cout << "Output is: "<< g;
 Return 0;
}
```

cin and strings

- The extraction operator (>>)can be used on cin to get strings of characters in the same way as with fundamental data types

```
string mystring;
cin >> mystring;
```

"Alice Wonderland"

  - However, cin extraction always considers whitespaces (spaces, tabs, new-line...) as terminating the value being extracted, and thus extracting a string means to always extract a single word, not a phrase or an entire sentence.
  - Whitespace is a term that refers to characters that are used for formatting purposes.

getline

- To read a string containing blanks, use the function getline
- The syntax to use the function getline is:

```
getline(istreamVar, strVar);
```

  - istreamVar is an input stream and strVar is a string variable. The reading is delimited by the newline character '\n'.
  - The function getline reads until it reaches the end of the current line. The newline character is also read but not stored in the string variable.

- Consider the following statement:

  ```
  string myString;

  getline(cin, myString);
  ```

- If the input is the following characters:

  “ Hello there. How are you?”

  - the value of myString is:
    - myString = " Hello there. How are you?"

cin and get

- The extraction operator (>>) skips all leading whitespace characters when scanning for the next input value.
  - whitespace characters:
    - Blanks(‘ ’), tabs(‘\t’), and newline (‘\n’)

```
Consider the variable declarations:
   char ch1, ch2;
   int num;
and the input:
    A 25

   cin >> ch1 >> ch2 >> num;
```

- The syntax of cin, together with the get function to read a character, follows:

```
cin.get(varChar);
```

  - varChar is a char variable to store the next input character

```
Consider the input again:
    A 25

cin.get(ch1);
cin.get(ch2);
cin >> num;
```

cin and fail

- cin.fail() or cin
  - detects whether the value entered fits the value defined in the variable.
- if cin.fail() is true, it means that
  - the entered value does not fit the variable
  - the variable will not be affected
  - the instream is still broken
  - the entered value is still in the buffer and will be used for the next "cin >> variable"statement.

cin and clean

- When an input stream enters the fail state, the system ignores all further I/O using that stream. You can use the stream function clear to restore the input stream to a working state.

- The syntax to use the function clearn is:

```
istreamVar.clear();
```

  - istreamVar is an input stream, such as cin.
  - For example, cin.clear(); variable

cin and ignore

- When you want to process only partial data (say, within a line), you can use the stream function ignore to discard a portion of the input.

- The syntax to use the function ignore is:

```
cin.ignore(intExp, chExp);
```

  - intExp is an integer expression yielding an integer value. It specifies the maximum number of characters to be ignored in a line.
  - chExp is a char expression yielding a char value.
  - For example, cin.ignore(100, '\n');

- cin.ignore(100, '\n’);
  - ignores either the next 100 characters or all characters until the newline character is found, whichever comes first.
- cin.ignore(100, 'A’);
  - ignoring the first 100 characters or all characters until the character 'A’ is found, whichever comes first.

- What value is assigned to a and b?

Consider the declaration:

```
int a, b;
```

and the input:

```
25 67 89 43 72
12 78 34
```

Now consider the following statements:

```
cin >> a;
cin.ignore(100, '\n');
cin >> b;
```

Formatting output

- The manipulators setprecision, fixed, scientific can be used for formatting output
- Header file `#include <iomanip>`
- Syntax:

```
setprecision(n)
```

  - n is the number of digits (counting from left to right, rounding may happen)
  - use the setprecision manipulator with cout and the insertion operator <<.
  - For example: `cout << setprecision(2);`
    - formats the output of decimal numbers to two digits ~~(decimal places)~~ until a similar subsequent statement changes the precision.

Stream manipulators

- are functions specifically designed to be used in conjunction with the insertion (<<) and extraction (>>) operators on stream objects
- Are used to change formatting parameters on streams and to insert or extract certain special characters.
- For example,
  - setprecision, fixed, scientific, showpoint, setw, setfill, left, right

Example

```
#include <iostream>
#include <iomanip>

int main () {
   double f =3.14159;
   std::cout << std::setprecision(5) << f << '\n';
   std::cout << std::setprecision(9) << f << '\n';
   return 0;
}
```

```
#include <iostream>
#include  <iomanip>
int main () {
  double f =3.14159;
  std::cout << std::setprecision(5) << f << '\n';
  std::cout << std::setprecision(9) << f << '\n';
  std::cout << std::fixed;
  std::cout << std::setprecision(5) << f << '\n';
  std::cout << std::setprecision(9) << f << '\n';
  return 0;
}
```

Fixed & scientific

- Manipulator fixed
  - output floating-point numbers in a fixed decimal format
  - Example

```
cout << fixed;
```

- Manipulator scientific
  - output floating-point numbers in scientific format.
  - Example

```
cout << scientific;
```

```
//Example: scientific and fixed
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  double hours = 35.45;
  double rate = 15.00;
  double tolerance = 0.01000;
  cout << setprecision(3);
  cout << "hours = " << hours << ", rate = " << rate << ", pay = " <<
hours * rate << ", tolerance = " << tolerance << endl << endl;
  cout << scientific;
  cout << "Scientific notation: " << endl;
  cout << "hours = " << hours << ", rate = " << rate << ", pay = " <<
hours * rate << ", tolerance = " << tolerance << endl << endl;
  cout << fixed;
  cout << "Fixed decimal notation: " << endl;
  cout << "hours = " << hours << ", rate = " << rate << ", pay = " <<
hours * rate << ", tolerance = " << tolerance << endl << endl;

  return 0;
}
```

- Manipulator showpoint
  - the output may not show the decimal point and the decimal part
  - To force output to show the decimal point and trailing zeros even if the decimal places are not needed.
  - The opposite of showpoint is noshowpoint.
- Examples
  - cout << showpoint;
  - cout << fixed << showpoint;

Force to show the decimal point and show six (6) valid digits

Force to show the decimal point and show six (6) valid digits after the decimal point

-
```
cout << expression or manipulator << expression or manipulator...;
```

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

int main () {
  double a = 30;
  double b = 10000.0;
  double pi = 3.1416;
  cout<< a << '\t' << b << '\t' << pi << '\n';
  cout<<showpoint<< a << '\t' << b << '\t' << pi << '\n';
  cout<<fixed<<showpoint<< a << '\t' << b << '\t' << pi << '\n';
  cout<<setprecision(7);
  cout <<showpoint << a << '\t' << b << '\t' << pi << '\n';
  cout <<noshowpoint << a << '\t' << b << '\t' << pi << '\n';
  return 0;
}
```

```
30      10000   3.1416
30.0000 10000.0 3.14160
30.000000       10000.000000    3.141600
30.0000000      10000.0000000   3.1416000
30.0000000      10000.0000000   3.1416000

--------------------------------
Process exited after 0.5471 seconds with return value 0
Press any key to continue . . .
```

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

int main () {
  double a = 30;
  double b = 10000.0;
  double pi = 3.1416;
  cout<< a << '\t' << b << '\t' << pi << '\n';
  cout<<showpoint<< a << '\t' << b << '\t' << pi << '\n';
  cout<<fixed<<showpoint<< a << '\t' << b << '\t' << pi << '\n';
  cout<<setprecision(7);
  cout <<showpoint << a << '\t' << b << '\t' << pi << '\n';
  cout <<noshowpoint << a << '\t' << b << '\t' << pi << '\n';
  return 0;
}
```

```
//Example: showpoint, scientific, and fixed
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  double hours = 35.45;
  double rate = 15.00;
  double tolerance = 0.01000;
  cout<<fixed<<showpoint;
  cout << setprecision(3);
  cout << "hours = " << hours << ", rate = " << rate << ", pay = " << hours *
rate << ", tolerance = " << tolerance << endl << endl;
  cout << scientific;
  cout << "Scientific notation: " << endl;
  cout << "hours = " << hours << ", rate = " << rate << ", pay = " << hours *
rate << ", tolerance = " << tolerance << endl << endl;
  cout << fixed;
  cout << "Fixed decimal notation: " << endl;
  cout << "hours = " << hours << ", rate = " << rate << ", pay = " << hours *
rate << ", tolerance = " << tolerance << endl << endl;

  return 0;
}
```

- Manipulator setw
  - output the value of an expression in a specific number of columns, that is, set field width
  - The output is right-justified and unused columns to the left are filled with spaces
- Syntax setw (n)
- For example:

```
double value=0.04;
Cout<<setw(8)<<value;
```

Additional Output Formatting Tools

- Additional formatting tools that give you more control over your output:
  - `setfill` manipulator
  - `left` and `right` manipulators
  - `unsetf` manipulator

Manipulator `setfill`

- Output stream variables can use **setfill** to fill unused columns with a character

```
ostreamVar << setfill(ch);
```

- Example:

```
cout << setfill('#') << setw(5);
cout << "ABC" << endl;
```

The output will be

```
##ABC
```

- The default character of **setfill** is the whitespace.

Manipulators `left` and `right`

- **`left`**: left-justifies the output

```
ostreamVar << left;
```

- **`right`**: right-justifies the output

```
ostreamVar << right;
```

- Example

```
cout << left;
cout << right;
```

Clear specific format flags

- Use **`unsetf`**
- Example:
  - Disable **`left`** by using **`unsetf`**

```
ostreamVar.unsetf(ios::left);
```

- For cout:

```
cout.unsetf(ios::left);
```

Types of Manipulators

- Two types of manipulators:
  - With parameters
  - Without parameters
- Parameterized: require `iomanip` header
  - `setprecision`, `setw`, and `setfill`
- Nonparameterized: require `iostream` header
  - `endl`, `fixed`, `showpoint`, `left`, and `flush`

Input/Output with files

- File: area in secondary storage to hold info
- File I/O is a five-step process
  - Include `fstream` header
  - Declare file stream variables
  - Associate the file stream variables with the input/output sources
  - Use the file stream variables with >>, <<, or other input/output functions
  - Close the files

Skeleton of file I/o programs

```
#include <fstream>
//Add additional header files you use
using namespace std;
int main()
{
   //Declare file stream variables such as the following
   ifstream inData;
   ofstream outData;
.
.
.
  //Open the files
  inData.open("progin.txt"); //open the input file
  outData.open("progout.txt"); //open the output file

  //Code for data manipulation

  //Close files
  inData.close();
  outData.close();

return 0;
}
```

Problem 1

- Q1: Write a program that reads a student name followed by five test scores. The program should output the student name, the five test scores, and the average test score. Output the average test score with two decimal places.
  - The data to be read is stored in a file called test.txt. The output should be stored in a file called result.txt.
  - Input A file containing the student name and the five test scores.
    - A sample input is:

```
Andrew Miller 87.50 89 65.75 37 98.50
```

  - Output The student name, the five test scores, and the average of the five test scores, saved to a file.
    - A sample out is

```
Student name: Andrew Miller
Test scores: 87.50 89.00 65.75 37.00 98.50
Average test score: 75.55
```


Solution
Analysis
Problem

Arrays and Strings

Dr. Eman Hammad
Eman.Hammad@tamuc.edu

In this lecture, we will learn

- Learn how to declare and initialize **arrays**
- Become familiar with the restrictions on array processing
- Use arrays and strings

Introduction

- Simple data type:
  - variables of these types can store only one value at a time
  - For example
    - int, double ,char, bool, short, long
- Structured data type:
  - a data type in which each data item is a collection of other data items

Arrays

- Array:
  - a collection of a fixed number of components, all of the same data type
- One-dimensional array:
  - components are arranged in a list form
  - Syntax for declaring a one-dimensional array:

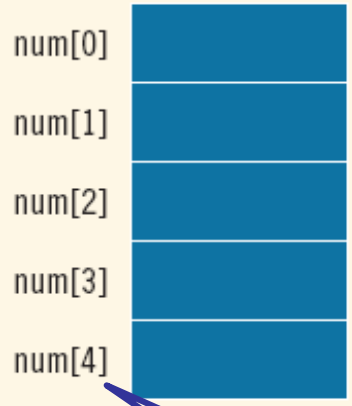
```
dataType arrayName[intExp];
```

    - `intExp`: any constant expression that evaluates to a positive integer
    - `[]` is the array subscripting operator

- Example:

Size of Array

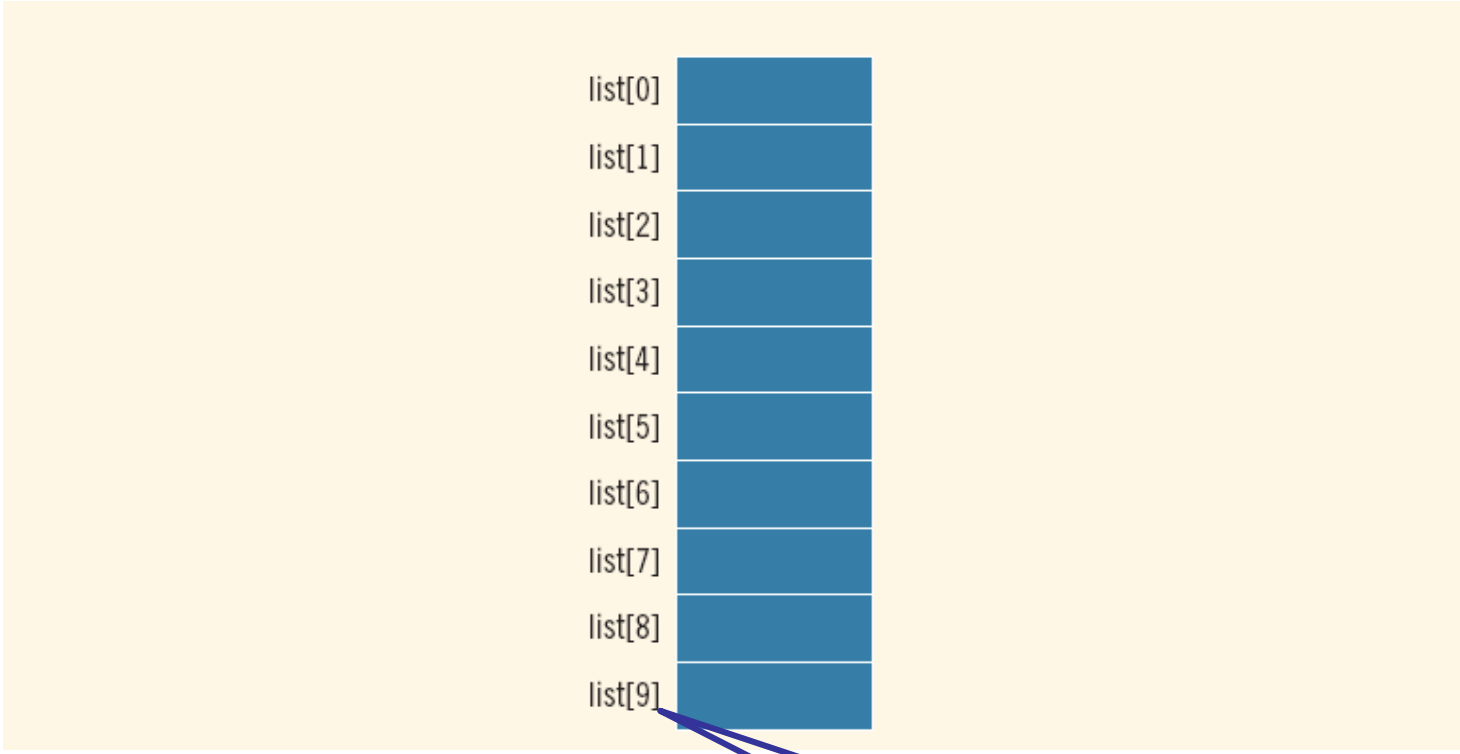
```
int num[5];
```



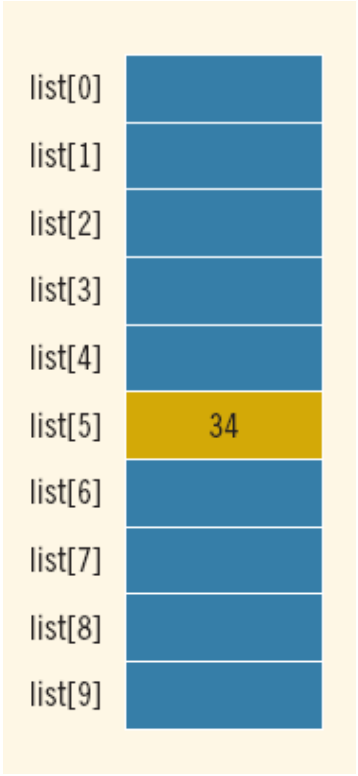
Index

Size of Array

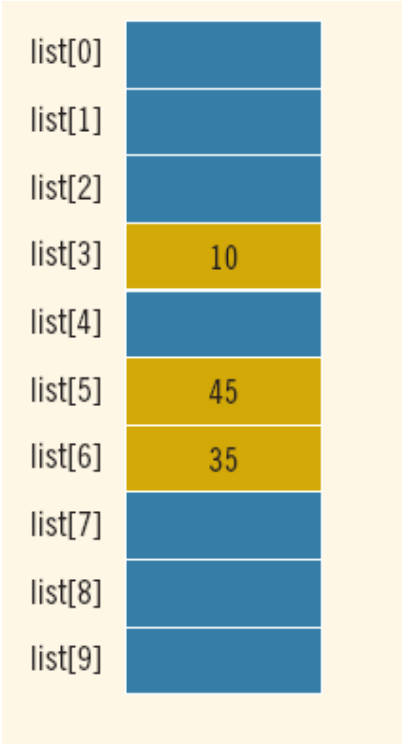
```
int list[10];
```



```
list[5] = 34;
```




```
list[3] = 10;
list[6] = 35;
list[5] = list[3] + list[6];
```



Processing One-Dimensional Arrays

- Basic operations on a one-dimensional array:
  - Initializing
  - Inputting data
  - Outputting data stored in an array
  - Finding the largest and/or smallest element
- Each operation requires ability to step through elements of the array
  - Easily accomplished by a loop

- Given the declaration:

```
int list[100];  //array of size 100
int i;
```

- Use a `for` loop to access array elements:

```
for (i = 0; i < 100; i++)          //Line 1
      cin >> list[i];              //Line 2
```

Array Index Out of Bounds

- Index of an array is in bounds
  - if the index is 0 <= index <= ARRAY_SIZE -1.
  - Otherwise, the index is out of bounds
  - In C++, there is no guard against indices that are out of bounds

- If we have the statements:

```
double num[10];
int i;
```

  - The component num[i] is valid if

```
i = 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9
```

Problem 1

- Create a C++ program that will
  - Declare an array named list of 100 components of type int
  - The values of all components are randomly generated numbers in the range [0,1000)
  - Print out all components
  - Find the sum and average of components in the array
  - Find the largest component in the array

More about Array Initialization

- Declaration and Initialization

```
int list[100];            //array of size 100
int i;
for (i = 0; i < 100; i++)
  list[i]=0;              //set to 0
```

- Initialization During Declaration

```
double sales[] = {12.25, 32.50, 16.90, 23, 45.68};
```

  - In this case, it is not necessary to specify the size of the array
    - Size determined by the number of initial values in the braces

- Partial Initialization of Arrays During Declaration

```
int list[10] = {0};
```

  - declares **list** to be an array of **10** components and initializes all of them to zero

```
int list[10] = {8,5,12};
```

  - declares **list** to be an array of **10** components, initializes `list[0]` to **8**, `list[1]` to **5**, `list[2]` to **12** and all other components are initialized to **0**

- More examples:

```
int list1[] = {5, 6, 3};
int list2[25]= {4, 7};
```

- Question:

What is the effect of above statements?

- Consider the following statements:

```
int myList[5] = {0, 4, 8, 12, 16}; //Line 1
int yourList[5]={100};             //Line 2

yourList=myList;
cin>>yourList;
cout<<yourList;
```

- What is the effect of above statements?

```
yourList=myList;   // illegal
cin>>yourList;     // illegal
cout<<yourList;    // legal but fails to give the desired results
```

Some Restrictions on Array Processing

- C++ does not allow aggregate operations on an array.
  - An aggregate operation on an array
    - is any operation that manipulates the entire array as a single unit.
- For instance:

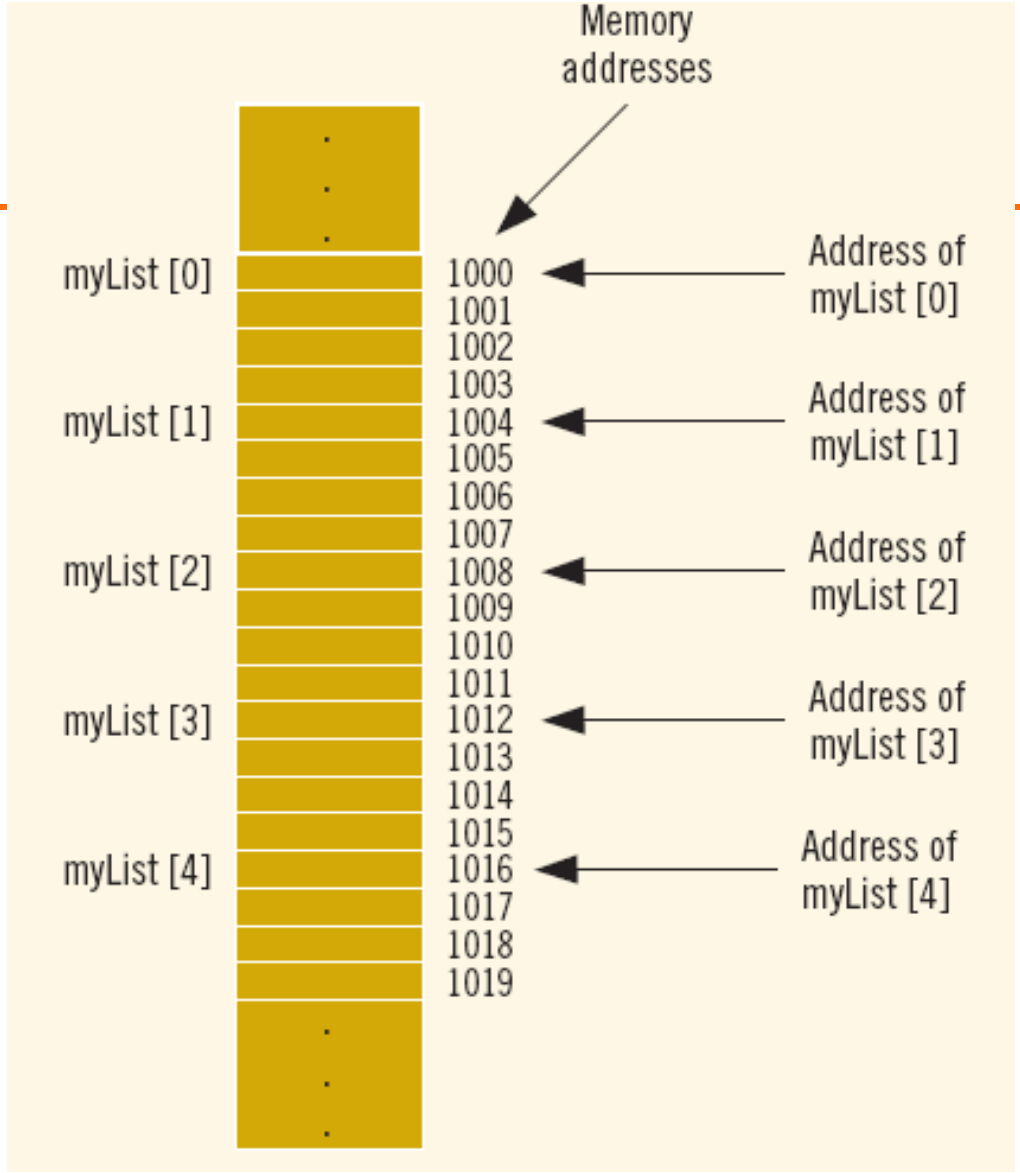
```
yourList=myList;   // illegal
cin>>yourList;     // illegal
cout<<yourList;    // legal but fails to give the desired results

if (myList <= yourList) //Base Address
...
```

Corrections:

```
int myList[5] = {0, 4, 8, 12, 16}; //Line
1 int yourList[5]={100};           //Line 2

yourList=myList; //illegal
//correction
yourList[0] = myList[0];
yourList[1] = myList[1];
yourList[2] = myList[2];
yourList[3] = myList[3];
yourList[4] = myList[4];
//or
for (int index = 0; index < 5; index++)
   yourList[index] = myList[index];

cin>>yourList; //illegal
//correction
for (int index = 0; index < 5; index++)
  cin >> yourList[index];

cout<<yourList; // legal but fails to give the desired results
//correction
for (int index = 0; index < 5; index++)
  cout<<yourList[index];
```

```
if (myList <= yourList) //Base Address
...
```

- myList or yourList
  - Refers to the base address of the array (the memory location)
  - That is the address of the component myList[0] or yourList[0]
- The statement
  - Evaluates to true if the base address of the array myList is less than the base address of the array yourList;
  - Otherwise, it evaluates to false


Memory addresses
myList [0]
myList [1]
myList [2]
myList [3]
myList [4]
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
Address of myList [0]
Address of myList [1]
Address of myList [2]
Address of myList [3]
Address of myList [4]

typedef in C++

- To make code more clear and easier to modify
- typedef
  - provides a way to declare an identifier as a type alias, to be used to replace a possibly complex type name or a previously defined data type
  - The general syntax of the typedef statement is:

```
typedef existingTypeName newTypeName;
```

    - In C++, typedef is a reserved word.
    - The typedef statement does not create any new data type; it only creates an alias to an existing data type.

- Examples:
  - The statement:

```
typedef int integer;
```

    - creates an alias, integer, for the data type int.
  - Similarly, the statement:

```
typedef double real;
```

    - creates an alias, real, for the data type double.
  - The statement:

```
typedef double decimal;
```

    - creates an alias, decimal, for the data type double.

Other Ways to Declare Arrays

```
const int SIZE = 50; //Line 1
typedef double list[SIZE]; //Line 2

list yourList; //Line 3
list myList; //Line 4
```

- Lines 3 and 4 are equivalent to:

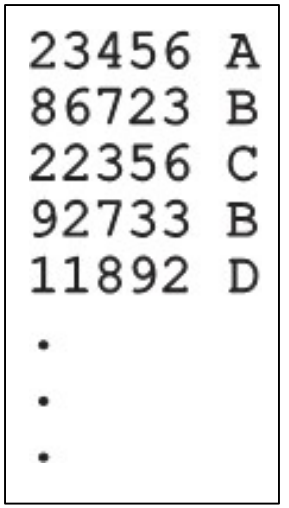
```
double yourList[50];
double myList[50];
```

Parallel Arrays

- Two (or more) arrays are called parallel if their corresponding components hold related information

- Example:

```
int studentId[50];
char courseGrade[50];
```



Two- and Multidimensional Arrays

- Two-dimensional array: collection of a fixed number of components (of the same type) arranged in two dimensions
  - Sometimes called matrices or tables
- Declaration syntax:

```
dataType  arrayName[intExp1][intExp2];
```

  - `intExp1` and `intExp2` are expressions with positive integer values specifying the number of rows and columns in the array

```
double sales[10][5];
```

Accessing Array Components

- Accessing components in a two-dimensional array:

```
arrayName[indexExp1][indexExp2]
```

  - Where `indexExp1` and `indexExp2` are expressions with positive integer values, and specify the row and column position

- Example:

```
sales[5][3] = 25.75;
```

Accessing Array Components (cont'd.)

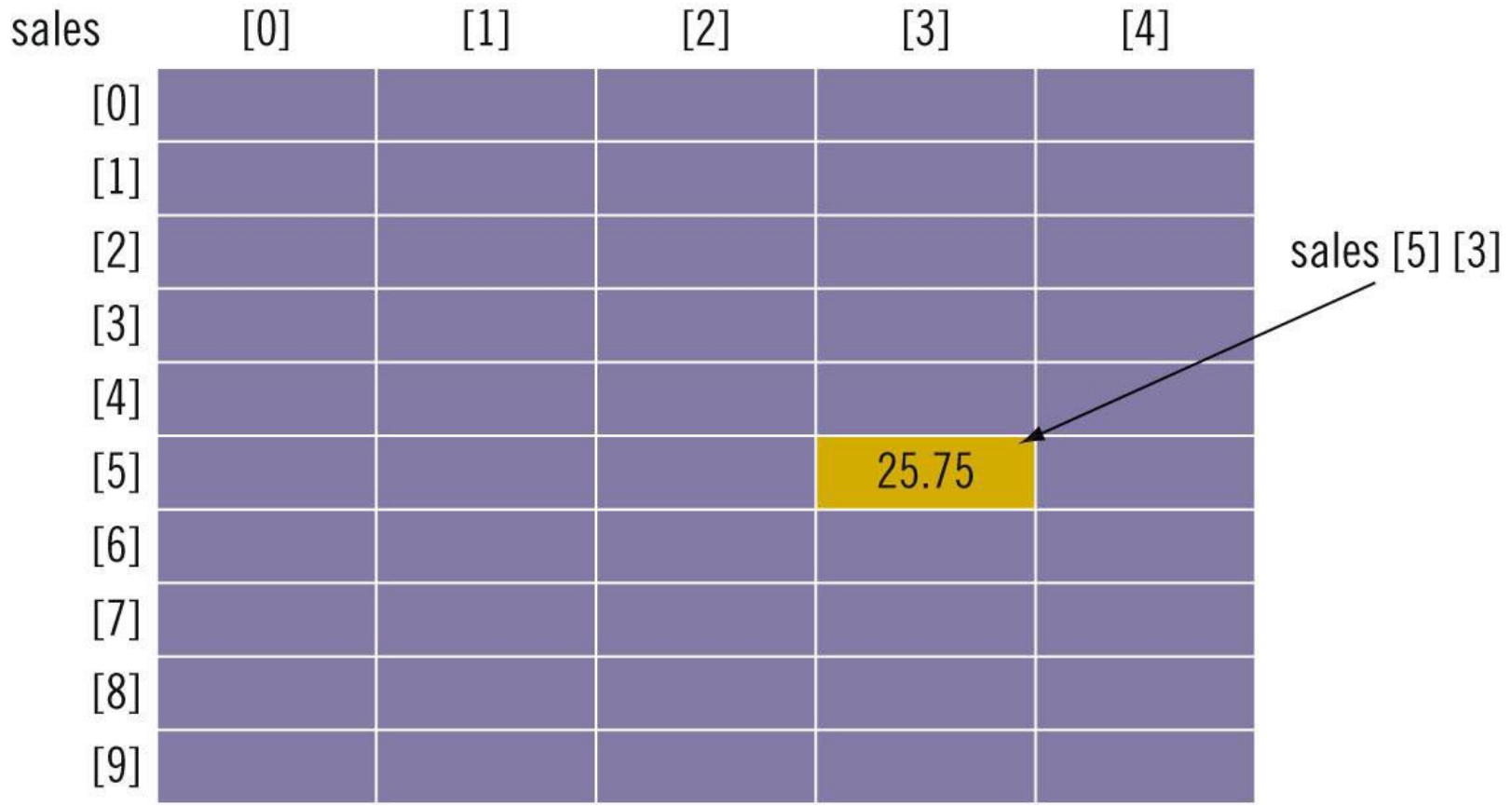


FIGURE 8-14 sales[5][3]

Two-Dimensional Array Initialization During Declaration

- Two-dimensional arrays can be initialized when they are declared:
  - Elements of each row are enclosed within braces and separated by commas
  - All rows are enclosed within braces
  - For number arrays, unspecified elements are set to 0

```
int board[4][3] = {{2, 3, 1},
                   {15, 25, 13},
                   {20, 4, 7},
                   {11, 18, 14}};
```

board	[0]	[1]	[2]
[0]	2	3	1
[1]	15	25	13
[2]	20	4	7
[3]	11	18	14

Processing Two-Dimensional Arrays

- Ways to process a two-dimensional array:
  - Process entire array
  - Row processing: process a single row at a time
  - Column processing: process a single column at a time
- Each row and each column of a two-dimensional array is a one-dimensional array
  - To process, use algorithms similar to processing one-dimensional arrays

Initialization

- Examples:
  - To initialize row number 4 (fifth row) to 0:

```
row = 4;
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
    matrix[row][col] = 0;
```

```
for (row = 0; row < NUMBER_OF_ROWS; row++)
    for (col = 0; col < NUMBER_OF_COLUMNS; col++)
        matrix[row][col] = 0;
```

Print

- Use a nested loop to output the components of a two dimensional array:

```
for (row = 0; row < NUMBER_OF_ROWS; row++)
{
    for (col = 0; col < NUMBER_OF_COLUMNS; col++)
        cout << setw(5) << matrix[row][col] << " ";

    cout << endl;
}
```

  - setw(5)
    - outputs the value of the next expression in 5 columns.
    - Is right-justified
    - First several columns can be left blank
    - setw controls the output of only the next expression.

Input

- Examples:
  - To input into the row number 4 (fifth row):

```
row = 4;

for (col = 0; col < NUMBER_OF_COLUMNS; col++)
    cin >> matrix[row][col];
```

  - To input data into each component of matrix:

```
for (row = 0; row < NUMBER_OF_ROWS; row++)
    for (col = 0; col < NUMBER_OF_COLUMNS; col++)
        cin >> matrix[row][col];
```

Sum by Row

- Example:
  - To find the sum of row number 4:

```
sum = 0;
row = 4;
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
    sum = sum + matrix[row][col];
```

Sum by Column

- Example:
  - To find the sum of each individual column:

```
  //Sum of each individual column
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
{
     sum = 0;
     for (row = 0; row < NUMBER_OF_ROWS; row++)
          sum = sum + matrix[row][col];

     cout << "Sum of column " << col + 1 << " = " << sum
          << endl;
}
```

2D C-Strings (2D Character Arrays)

```
char list[100][16];
strcpy(list[1], "Snow White");
```

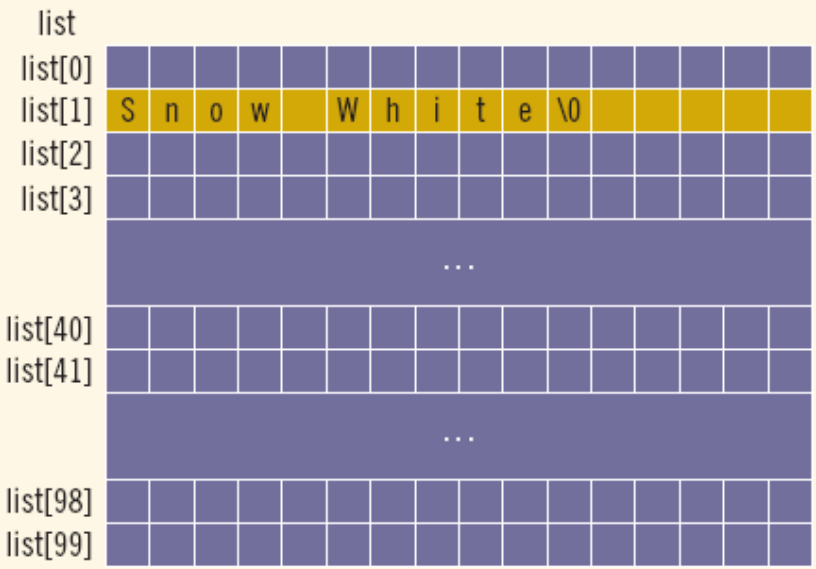


FIGURE 9-17 Array `list`, showing `list[1]`

```
for (j = 0; j < 100; j++)
    cin.get(list[j], 16);
```

http://codepad.org/Owx36eXS

Another Way to Declare a Two-Dimensional Array

- Can use **typedef** to define a two-dimensional array data type:

```
const int NUMBER_OF_ROWS = 20;
const int NUMBER_OF_COLUMNS = 10;

typedef int tableType[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
```

- To declare an array of 20 rows and 10 columns:

```
tableType matrix;
```

Multidimensional Arrays

- *n*-dimensional array: collection of a fixed number of elements arranged in *n* dimensions (*n* >= 1)

- Declaration syntax:

```
dataType arrayName[intExp1][intExp2] ... [intExpn];
```

- To access a component:

```
arrayName[indexExp1][indexExp2] ... [indexExpn]
```

C-String  String

C-Strings (Character Arrays)

- Character arrays in C++are of special interest, and you process them differently than you process other arrays

- Character array (**a.k.a. null-terminated string**):
  - an array whose components are of type `char`

```
char name[16];
```

- C-strings are null-terminated (`'\0'`) character arrays
  - Example:
    - `'A'` is the character `A`
    - `"A"` is the C-string `A`
      - `"A"` represents two characters, `'A'` and `'\0'`, where `'\0'` is the null character

- Example: `char name[5];`
  - Since C-strings are null terminated and name has 5 components, the largest string it can store has 4 characters
  - If you store a string whose length is less than the array size, the last components are unused
  - Array elements after the null character are not part of the string, and their contents are irrelevant.

- Consider the following statements:

```
char name[5]= "ABCDE";
int n= sizeof(name)/sizeof(char);
cout<<n;
```

Program: http://codepad.org/vut0B1ku

- Size of an array can be omitted if the array is initialized during declaration

- Example:

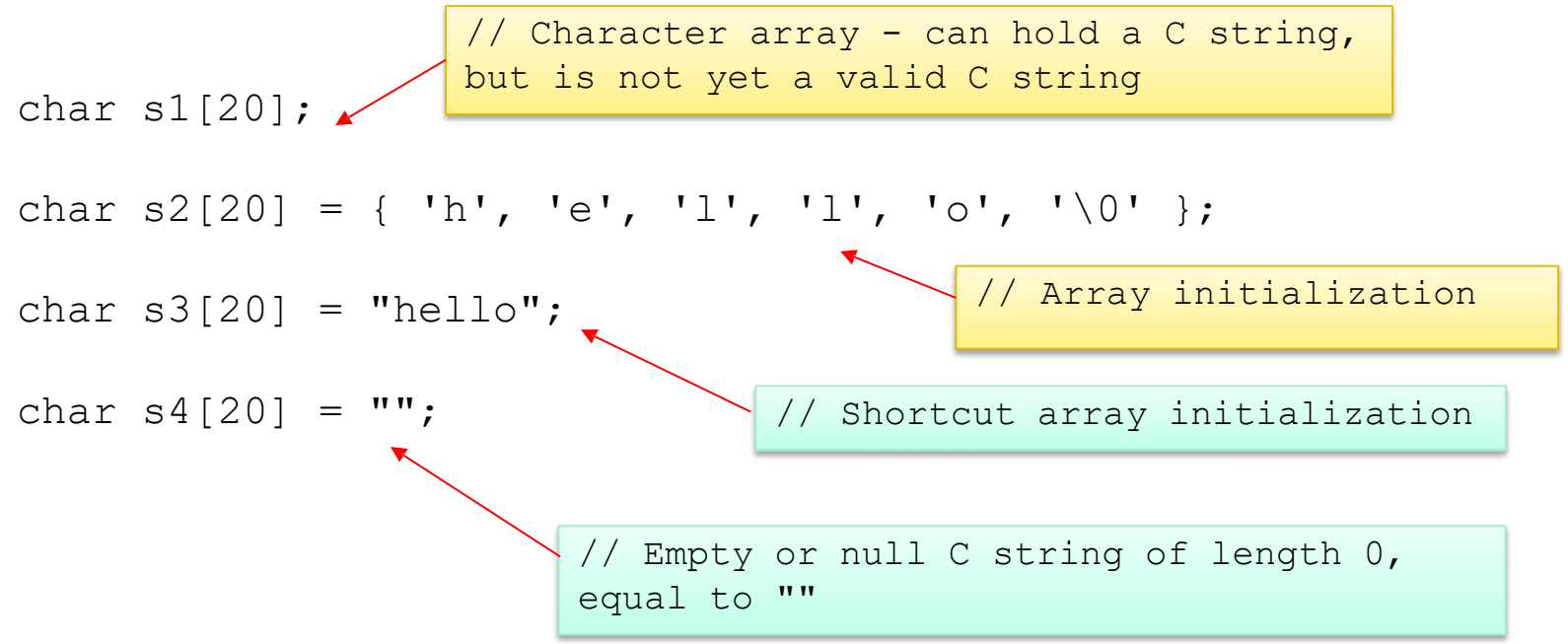
```
char name[] = "John";
```

  - Declares an array of length 5 and stores the C-string "John" in it

Program: http://codepad.org/3Cod1D9m

- Useful string manipulation functions
  - strcpy, strcmp, and strlen
  - #include <cstring>

- Here are some examples of declaring C strings as arrays of char:



- Given

```
char s3[10] = "hello";
```

  - The following diagram shows how s3 is represented in memory

    - terminated by a null character. Array elements after the null character are not part of the array
    - strlen(s3) // can get the number of valid characters in the array (not including the null character)

```
char s1[20];

char s2[20] = { 'h', 'e', 'l', 'l', 'o', '\0' };

char s3[20] = "hello";

char s4[20] = "";
```

- What if we have the following statements,

```
s1[20]= "hello"; //statement 1

s1=s3; //statement 2
```

  - What is the value of the array s1?

Reading and Writing Strings

- Most rules for arrays also apply to C-strings (which are character arrays)

- Attention:
  - Aggregate operations, such as assignment and comparison, are not allowed on arrays

```
cin>>yourList; //illegal
```

  - C++ does allow aggregate operations for the input and output of C-strings

```
char name[5];
cin>>name; //legal
```

C-string Input

- Example:

```
char name[5];
cin>>name; //legal
```

  - Stores the next input C-string into name

- To read strings with blanks, use `get` function:

```
char str[31];
cin.get(str, 31);
```

  - Stores the next `30` characters into `str` but the newline character is not stored in `str`
  - If input string has fewer than `30` characters, reading stops at the newline character

```
char str[31];
cin.get(str, 31);
```

- Suppose that the input is:

  " Hello there. My name is Mickey Mouse’’

- Then, because str can store, at most, 30 characters, the C-string

  "Hello there. My name is Mickey" is stored in str.

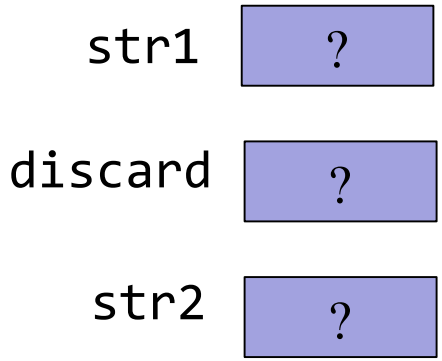
Why not 31?

- Now, suppose that we have the statements:

```
char str1[26];
char str2[26];
char discard;

cin.get(str1,26);
cin.get(discard);
cin.get(str2,26);
```

  - and the two lines of input:

```
Summer is warm.
Winter will be cold.
```



Program: http://codepad.org/jX5RuxvK

String Output

- Example:

```
cout<<str1<<endl;
cout<<discard<<endl;
cout<<str2<<endl;
```

  - Outputs the content of str, discard, and str2 on the screen
  - << continues to write the contents of name until it finds the null character('\0', called nul in ASCII)
  - If does not contain the null character, then strange output may occur
    - << continues to output data from memory adjacent to until a '\0' is found

Specifying Input/Output Files at Execution Time

- User can specify the name of an input and/or output file at execution time:

- Example：
  - Consider the following statements:

```
#include<fstream>
ifstream infile;
ofstream outfile;
char fileName[51];
```

  - To allow the user to specify the input and output files, we can have

```
cout << "Enter the input file name: ";
cin >> fileName;
infile.open(fileName); //open the input file
// data processing
cout << "Enter the output file name: ";
cin >> fileName;
outfile.open(fileName); //open the output file
```

Input/Output with files (Review)

- File: area in secondary storage to hold info
- File I/O is a five-step process
  - Include `fstream` header
  - Declare file stream variables
  - Associate the file stream variables with the input/output sources
  - Use the file stream variables with >>, <<, or other input/output functions
  - Close the files

Skeleton of file I/o programs (Review)

```
#include <fstream>
//Add additional header files you use
using namespace std;
int main()
{
   //Declare file stream variables such as the following
   ifstream inData;
   ofstream outData;
.
.
.
  //Open the files
  inData.open("progin.txt"); //open the input file
  outData.open("progout.txt"); //open the output file

  //Code for data manipulation

  //Close files
  inData.close();
  outData.close();

return 0;
}
```

Problem 1

- Q1: Write a program that reads a student name followed by five test scores. The program should output the student name, the five test scores, and the average test score. Output the average test score with two decimal places.
  - The data to be read is stored in a file called test.txt. The output should be stored in a file called result.txt.
  - Input A file containing the student name and the five test scores.
    - A sample input is:

```
Andrew Miller 87.50 89 65.75 37 98.50
```

  - Output The student name, the five test scores, and the average of the five test scores, saved to a file.
    - A sample out is

```
Student name: Andrew Miller
Test scores: 87.50 89.00 65.75 37.00 98.50
Average test score: 75.55
```

Solution (Review)

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>

using namespace std;

int main()
{
    //Declare variables;                          Step 1
    ifstream inFile; //input file stream variable
    ofstream outFile; //output file stream variable

    double test1, test2, test3, test4, test5;
    double average;

    string firstName;
    string lastName;

    inFile.open("test.txt");                        //Step 2
    outFile.open("testavg.txt");                    //Step 3

    outFile << fixed << showpoint;                 //Step 4
    outFile << setprecision(2);                    //Step 4

    cout << "Processing data" << endl;
    //next slide
```

```
    cout << "Processing data" << endl;

    inFile >> firstName >> lastName;                //Step 5
    outFile << "Student name: " << firstName
            << " " << lastName << endl;             //Step 6

    inFile >> test1 >> test2 >> test3
           >> test4 >> test5;                       //Step 7
    outFile << "Test scores: " << test1
            << " " << test2 << " "  << test3
            << " "  << test4 << " "  << test5
            << endl;                                //Step 8

    average = (test1 + test2 + test3 + test4
              + test5) / 5.0;                       //Step 9

    outFile << "Average test score: " << " "
            << average << endl;                     //Step 10

    inFile.close();                                 //Step 11
    outFile.close();                                //Step 11

    return 0;
}
```

Program at http://codepad.org/9eKMgC8L

Specifying Input/Output Files at Execution Time

- User can specify the name of an input and/or output file at execution time:

- Example
  - Consider the following statements:

```
#include<fstream>
ifstream inFile;
ofstream outFile;
char fileName[51];
```

  - To allow the user to specify the input and output files, we can have

```
cout << "Enter the input file name: ";
cin >> fileName;
inFile.open(fileName); //open the input file
// data processing
cout << "Enter the output file name: ";
cin >> fileName;
outFile.open(fileName); //open the output file
```

Skeleton of file I/o programs with C-strings

```
#include <fstream>
//Add additional header files you use
using namespace std;
int main()
{
   ifstream inFile;
   ofstream outFile;
   char fileName[51];

  //Open the files
  //inFile.open("progin.txt"); //open the input file
  //outFile.open("progout.txt"); //open the output file
   cout << "Enter the input file name: ";
   cin >> fileName;
   inFile.open(fileName); //open the input file
    // data processing
   cout << "Enter the output file name: ";
   cin >> fileName;
   outFile.open(fileName); //open the output file

  //Code for data manipulation

  //Close files
  inFile.close();
  outFile.close();

return 0;
}
```

- Rewrite the solution
  - Using cin.eof() or inFile.eof()
    - cin and inFile are input stream variables
    - cin.eof() and inFile.eof() are logical (Boolean) expression
      - If the program has reached the end of the input data,
        - the input stream variable (cin and inFile)returns the logical value false.
        - The expressions are false

Program at http://codepad.org/zctw5Fjy

C-Strings VS. Strings

```
//C-Strings (Character Arrays)
 char cstr1[]="Hello";//csting-null terminated
 char cstr2[]={'H','e','l','l','o','\0'};

 char cstr3[6];
 //cstr3="Hello"; // invalid
 //cstr3={'H','e','l','l','o','\0'};// invalid

 strcpy(cstr3,"Hello"); // correction or
 cin>>cstr3; // only can hold 5 characters
 cin.get(cstr3,6);
 for(int i=0;i<5;i++)
    cin>>cstr3[i];
 cstr3[5]='\0';

  //the size of the array
  int n=sizeof(cstr1)/sizeof(char);
  //the number of characters excluding
  int m=strlen(cstr2);

  //output
  cout<<cstr1<<" "<<cstr2<<" "<<cstr3<<endl;
  for(int i=0;i<6;i++)
   cout<<cstr3[i];
```

http://codepad.org/qIEGvNWC

```
 //Strings (class) has a set of redefined methods
string str1="Hello";
string str2={'H','e','l','l','o','\0'}; //valid in c++11

string str3;
str3="Hello"; // valid
str3={'H','e','l','l','o','\0'};// valid in c++11

//input
cin>>str3; // stop reading when meeting spaces
(whitespaces, tabs, new-line...)
cin.ignore(100,'\n'); // or char ch; cin.get(ch);
//reads until it reaches the end of the current line
getline(cin,str3);

//the size of the string (i.e.,the number of characters)
int n=str1.size();
int m=str1.length();
cout<<n <<" "<<m<<endl;

//output
cout<<str1<<" "<<str2<<" "<<str3<<endl;
for(int i=0;i<6;i++)
    cout<<str3[i];
```

http://codepad.org/N4FbQ4iB

Input methods

```
char ch;
char cstr[10];
string str;

cin>>ch; // reads in any non-blank char
cin.get(ch); //reads in any char
cin>>cstr; //reads non-blank characters
cin.get(cstr,10); //reads in a C-string that may contain blanks,
ensures<=9 chars are read in
cin.ignore(); // skips over next char in the input buffer
cin.ignore(100,'\n');//skip next 100 characters or
// all characters until the newline character is found,
// whichever comes first.
//cin.fail(); //check if the input stream fails
//cin.eof();// tests for end-of-file
cin.clear(); //clears the error flag on cin
cin>>str; // reads in a string with no blanks
getline(cin,str); //reads in a string that may contain blanks
```

http://codepad.org/h0FRSlgQ

String to C-string

- Syntax:

```
strVar.c_str()
```

  - Where strVar is a variable of type string

- Example:

```
string str1="Hello";
str1.c_str();// a c-string
```

- Arrays of Strings

```
string list[100];
for(int i=0;i<100;i++)
  getline(cin,list[i]);
```

String Comparison (review)

- C-strings are compared character by character using the collating sequence of the system

- If we are using the ASCII character set
  - `"Air" < "Boat"`
  - `"Air" < "An"`
  - `"Bill" < "Billy"`
  - `"Hello" < "hello"`

```
#include <cstring>
```

Function	Effect
`strcpy(s1, s2)`	Copies the string `s2` into the string variable `s1` The length of `s1` should be at least as large as `s2`
`strcmp(s1, s2)`	Returns a value < `0` if `s1` is less than `s2` Returns `0` if `s1` and `s2` are the same Returns a value > `0` if `s1` is greater than `s2`
`strlen(s)`	Returns the length of the string `s`, excluding the null character

- Consider following statements:

```
char studentName[21];
char myname[16];
char yourname[16];
```

- The following statements

Statement	Effect
`strcpy(myname, "John Robinson");`	
`strlen("John Robinson");`	
`int len;` `len = strlen("Sunny Day");`	

Program: http://codepad.org/ZiOKQN44

- Consider following statements:

```
char studentName[21];
char myname[16];
char yourname[16];
```

- The following statements

Statement	Effect
`strcpy(myname, "John Robinson");`	`myname = "John Robinson"`
`strlen("John Robinson");`	Returns 13, the length of the string `"John Robinson"`
`int len;` `len = strlen("Sunny Day");`	Stores 9 into `len`

Program: http://codepad.org/ZiOKQN44

Statement

```
strcpy(myname, "John Robinson");

strlen("John Robinson");


int len;
len = strlen("Sunny Day");

strcpy(yourname, "Lisa Miller");
strcpy(studentName, yourname);

strcmp("Bill", "Lisa");

strcpy(yourname, "Kathy Brown");
strcpy(myname, "Mark G. Clark");
strcmp(myname, yourname);
```

Effect

Statement	Effect
`strcpy(myname, "John Robinson");`	`myname = "John Robinson"`
`strlen("John Robinson");`	Returns 13, the length of the string `"John Robinson"`
`int len;` `len = strlen("Sunny Day");`	Stores 9 into `len`
`strcpy(yourname, "Lisa Miller");` `strcpy(studentName, yourname);`	`yourname = "Lisa Miller"` `studentName = "Lisa Miller"`
`strcmp("Bill", "Lisa");`	Returns a value < 0

`strcpy(yourname, "Kathy Brown");`
`strcpy(myname, "Mark G. Clark");`
`strcmp(myname, yourname);`
	`yourname = "Kathy Brown"`
	`myname = "Mark G. Clark"`
	Returns a value > 0

Exercises

- Q1:What would be a valid range for the index of an array of size 99?
- What is the index of the first element?
- What is the index of the last element?

- Q2:what is stored in list after the following C++ code executes?

```
int list[8];

list[0] = 1;
list[1] = 2;
for (int i = 2; i < 8; i++)
{
    list[i] = list[i - 1] *  list[i - 2];
    if (i > 5)
        list[i] = list[i] - list[i - 1];
}
```

- Q3: Suppose that you have the following declaration:

```
int list[7] = {6, 10, 14, 18, 22};
```

- If this declaration is valid, what is stored in each of the eight components of list.

- Q4: Given the declaration:

```
char str1[21];
char str2[21];
```

  - Write a C++ statement that stores "Sunny Day" in str1.
  - Write a C++ statement that stores the length of str1 into the int variable length.
  - Write a C++ statement that copies the value of name into str2.
  - Write C++ code that outputs str1 if str1 is less than or equal to str2, and otherwise outputs str2.