

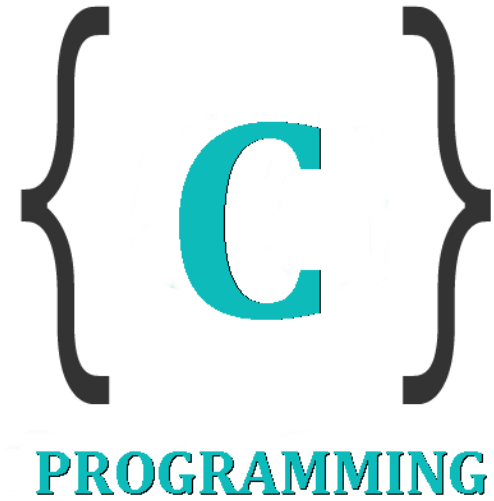
C Programming #1

Getting Started in C Programming

International College, KMITL

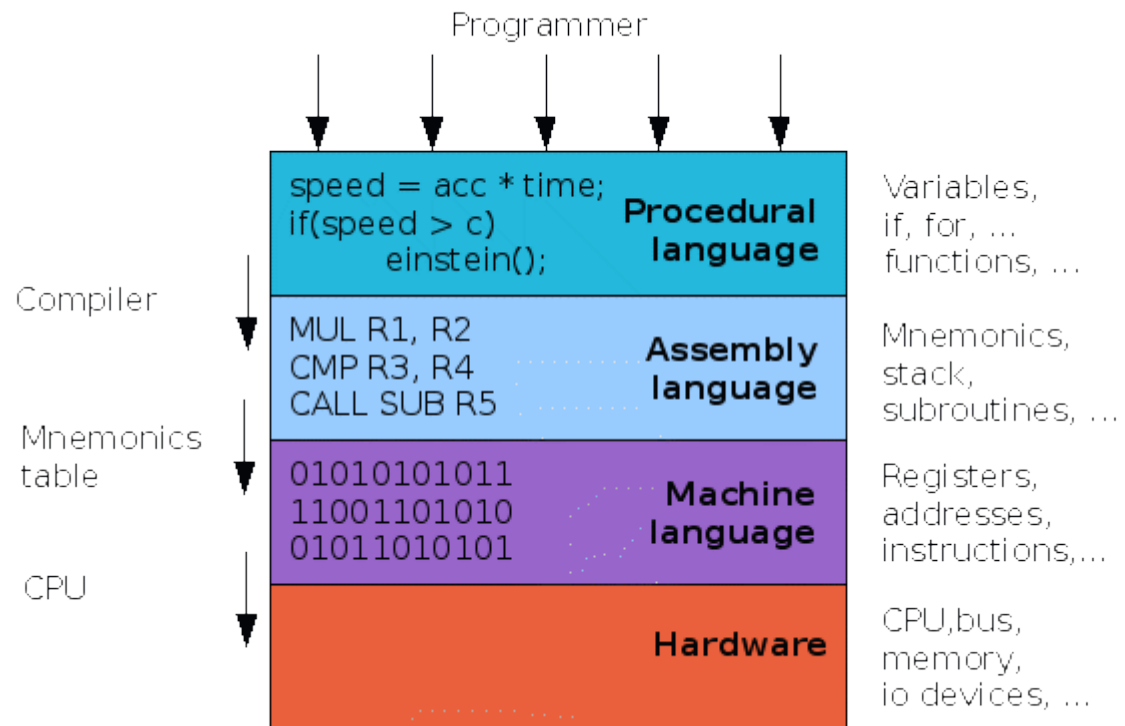
What is C Programming?

C is a high-level structured language



Structured language

Structured language is a high-level procedural language that enforces structured procedures



Interpreter & Compiler

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Compiler & Interpreter (continued)



Figure: Compiler



Figure: Interpreter

Integrated development environment (IDE)

An **integrated development environment (IDE)** is a programming environment that has been packaged as an application program, typically consisting of a code editor, a compiler, a debugger, and a graphical user interface (GUI) builder. e.g. code::blocks, Microsoft visual studio, Xcode, Pycharm, etc.



Algorithm

Algorithm is a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

algorithm
noun

Word, used by programmers
When they do not want to
Explain what they did.

Algorithm of Success

```
while(noSuccess)
{
    tryAgain();
    if(Dead) break;
}
```

C Programming structure

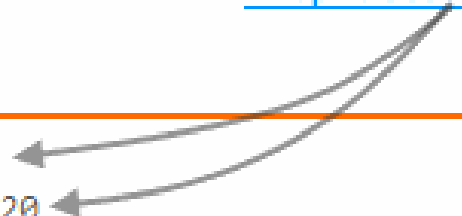
- ▶ Preprocessor directives
- ▶ Global declaration
- ▶ main() Function

```
#include <stdio.h> → Preprocessor directive  
Global declaration  
main() → Main function  
  
{  
    printf("hello, world\n");  
}
```


Preprocessor Directive

This part modifies a source file before handing it over to the compiler. This part is normally contribute to calling required header and define constant value.

Preprocessor Directives



```
#include <stdio.h>
#define MAX_FILES 20

void main(){
    printf("Maximum no. of files = %d.", MAX_FILES);
}
```

Header Files

Header Files are files that contain specific set of functions that are going to be invoked in the program.

List of Some Commonly used header Files and their purpose		
Header Files	Purpose	Functions declared
Stdio.h	Used for standard input and Output (I/O) operations.	printf(),scanf(),getchar(), putchar(), gets(), puts(), getc(), putc(), fopen, fclose, feof()
Conio.h	Contains declaration for console I/O function	clrscr(), getch(), exit()
Ctype.h	Used for character-handling or testing character.	isupper(),islower,isalpha()
Math.h	Declares mathematical functions and macros.	pow(), sqrt(), cos(), tan(), sin(), log()
Stdlib.h	Used for number conversions, storage allocations.	rand(), srand()
String.h	Used for manipulating strings.	strlen(),strctp(), strcmp(), strcat(), strlwr(),strupr(), strev()
Time.h	Used for manipulating time and date.	

Identifiers

Identifiers in C consist of three types:

- ▶ Reserved words
- ▶ Standard identifiers
- ▶ Programmer-created identifiers

Identifiers (continued)

- **Reserved word**: word that is predefined by the programming language for a special purpose and can only be used in a specified manner for its intended purpose - Also referred to as keywords in C

Table 2.1 Keywords					
auto	default	float	register	struct	volatile
break	do	for	return	switch	while
case	double	goto	short	typedef	
char	else	if	signed	union	
const	enum	int	sizeof	unsigned	
continue	extern	long	static	void	

Identifiers (continued)

- ▶ **Standard identifiers** are words predefined in C.
- ▶ Most of the standard identifiers are the names of functions that are provided in the C standard library
- ▶ It is good programming practice to use standard identifiers only for their intended purpose

Table 2.2 Sample of C Standard Identifiers

abs	fopen	isalph	rand	strcpy
argc	free	malloc	rewind	strlen
argv	fseek	memcpy	scanf	tolower
calloc	gets	printf	sin	toupper
fclose	isascii	puts	strcat	ungetc

Identifiers (continued)

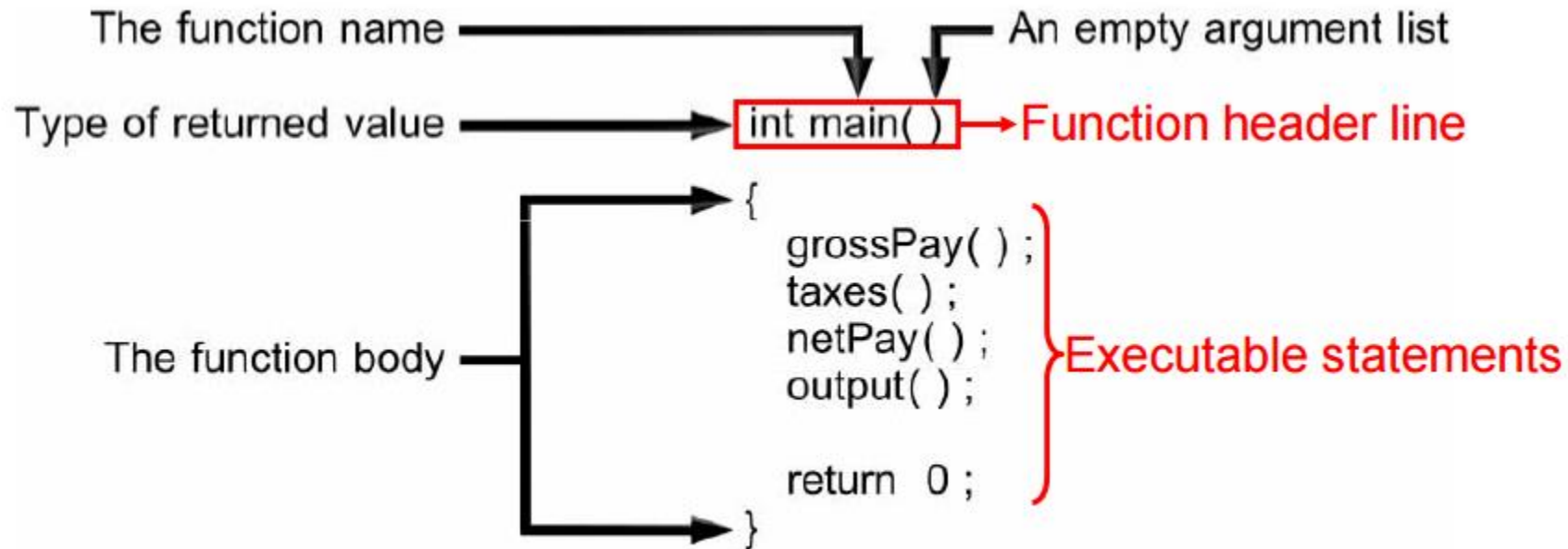
Programmer-created identifiers: selected by the programmer

- ▶ Also called programmer-created names
- ▶ Used for naming data and functions
- ▶ Must conform to C's identifier
- ▶ Can be any combination of letters, digits, or underscores (_) subject to the following rules:
 - First character must be a letter or underscore (_)
 - Only letters, digits, or underscores may follow the initial character
 - Blank spaces are not allowed
 - Cannot be a reserved word

Identifiers (continued)

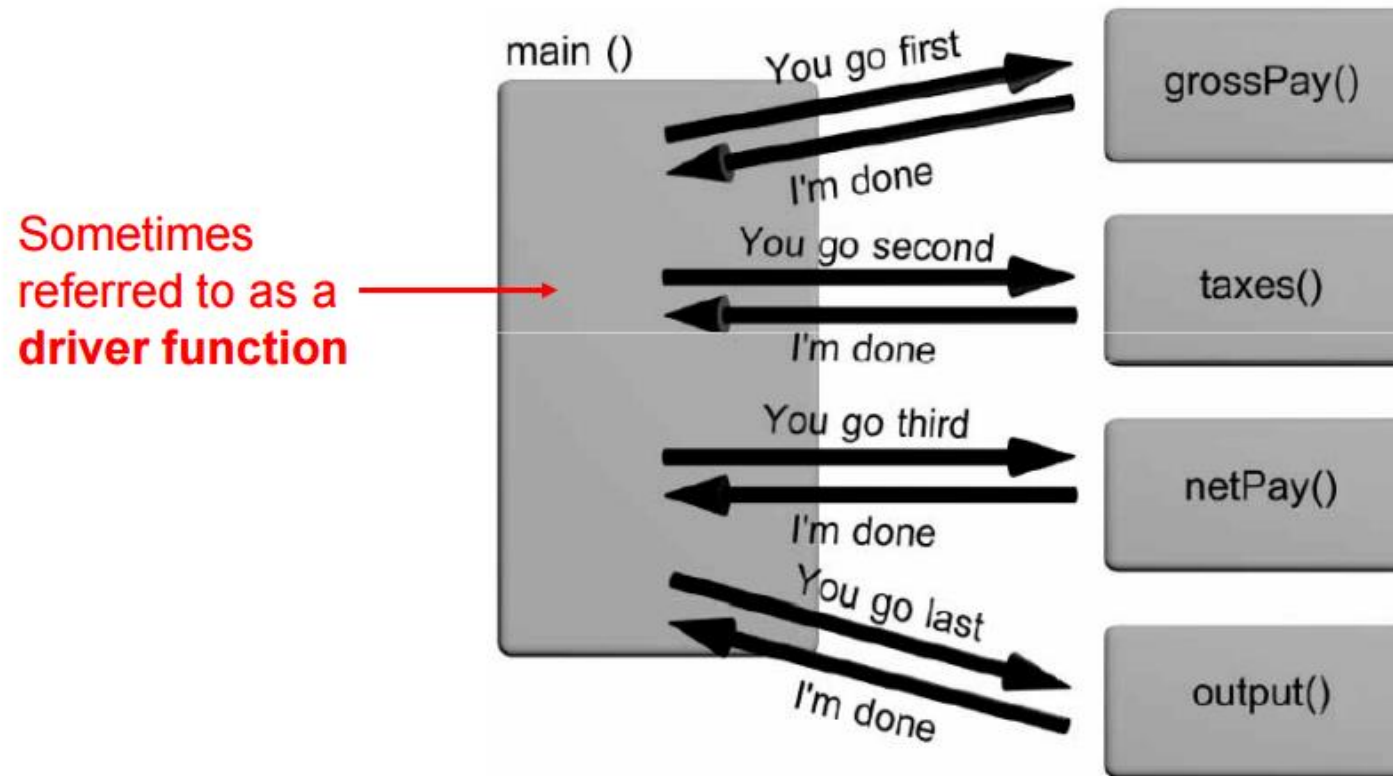
- ▶ Examples of invalid C programmer-created names:
 - 4ab7
 - calculate total
 - while
- ▶ All uppercase letters used to indicate a constant
- ▶ A function name must be followed by parentheses
- ▶ An identifier should be descriptive: degToRadians()
 - Bad identifier choices: easy, duh, justDolt
- ▶ C is a case-sensitive language
 - TOTAL, and total represent different identifiers

The main() Function



A sample main() function

The main() Function (continued)

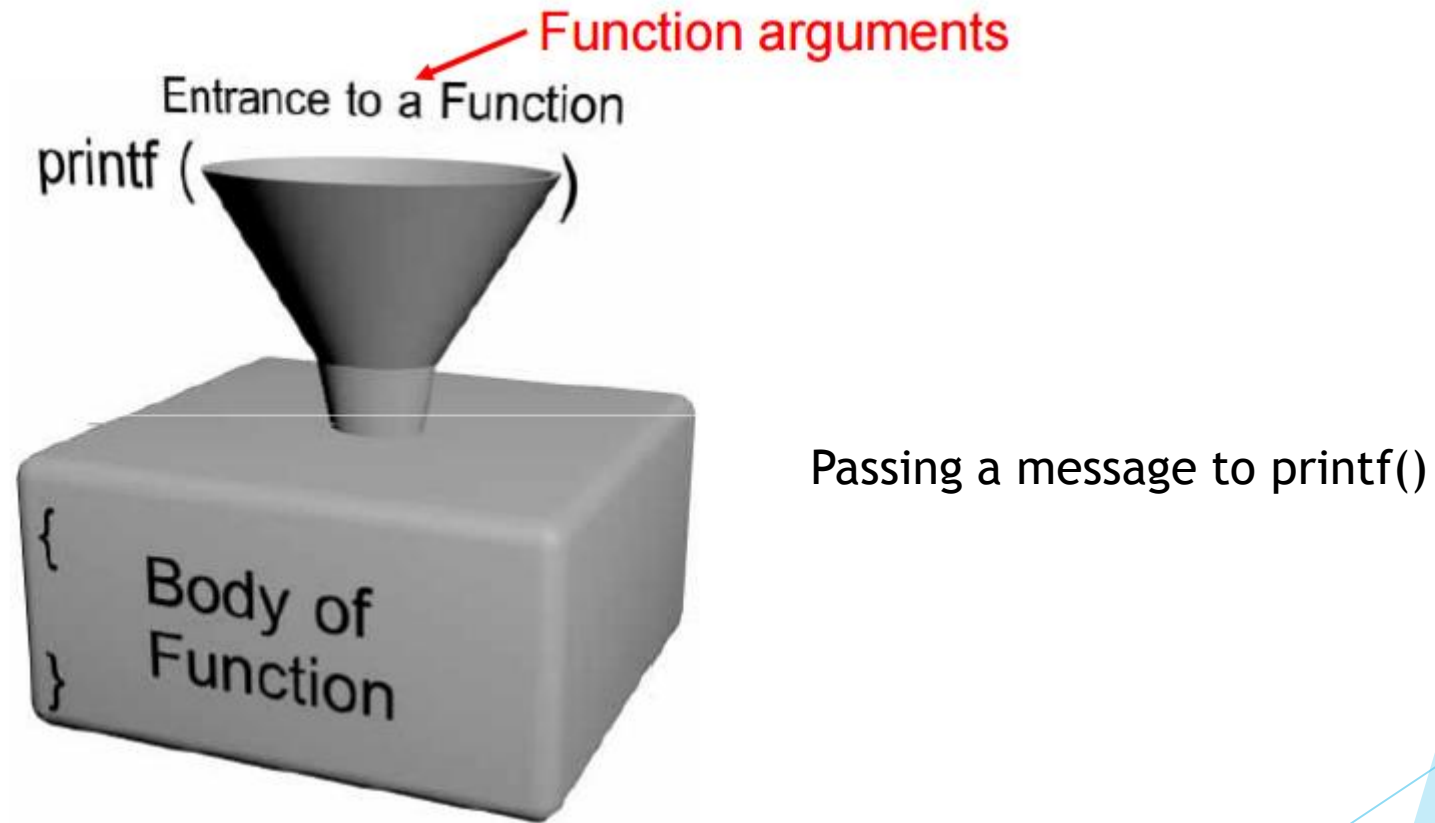


The `main()` function controls all other functions

The printf() Function

- ▶ printf() formats data and sends it to the standard system display device (i.e., the monitor)
- ▶ Inputting data or messages to a function is called passing data to the function
 - `printf("Hello there world!");`
- ▶ Syntax: set of rules for formulating statements that are “grammatically correct” for the language
- ▶ Messages are known as strings in C
 - `printf("Hello there world!");`

The printf() Function (continued)



system(“pause”)

system(“pause”) is a function used for stopping the console window.

- ▶ Normally used in Microsoft Visual Studio

Check Point #1

Programming Style: Indentation

- ▶ Except for strings, function names, and reserved words, C ignores all white space
- ▶ In standard form:
 - A function name is placed, with the parentheses, on a line by itself starting at the left-hand corner
 - The opening brace follows on the next line, under the first letter of the function name
 - The closing function brace is placed by itself at the start of the last line of the function

```
#include <stdio.h>

int main()
{
    printf("Hello, World");

    system("pause");
    return 0;
}
```

Programming Style: Indentation (continued)

- ▶ Within the function itself, all program statements are indented two spaces
- ▶ Don't do this:

```
#include <stdio.h>
int
main(
)
{printf("Hello, World");
system("pause")
;return 0;}
```

Programming Style: Comments

- ▶ Comments help clarify what a program does, what a group of statements is meant to accomplish, etc.
- ▶ The symbols `/*`, with no white space between them, designate the start of a comment; the symbols `*/` designate the end of a comment
- ▶ Comments can be placed anywhere within a program and have no effect on program execution
- ▶ Under no circumstances may comments be nested

```
#include <stdio.h>

/* this comment is /* always */ invalid * /
int main()
{
    printf("Hello, World");

    system("pause");
    return 0;
}
```



```
#include <stdio.h>

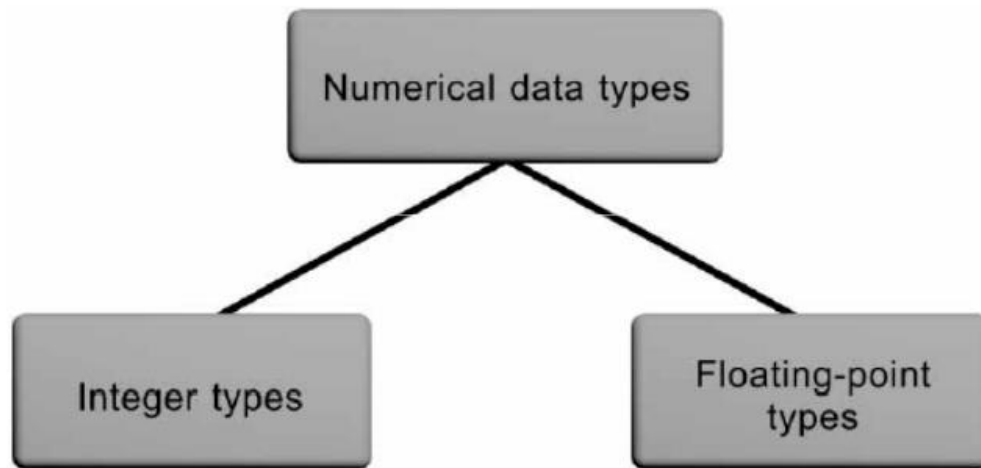
/* This is a comment*/
int main()
{
    printf("Hello, World");

    system("pause");
    return 0;
}
```



Data Types

- ▶ Data type: set of values and a set of operations can be applied to these values
- ▶ Built-in data type: is provided as an integral part of the language; also known as primitive type
- ▶ A literal is an acceptable value for a data type - Also called a literal value or constant - 2, 3.6, -8.2, and "Hello World!" are literal values because they literally display their values



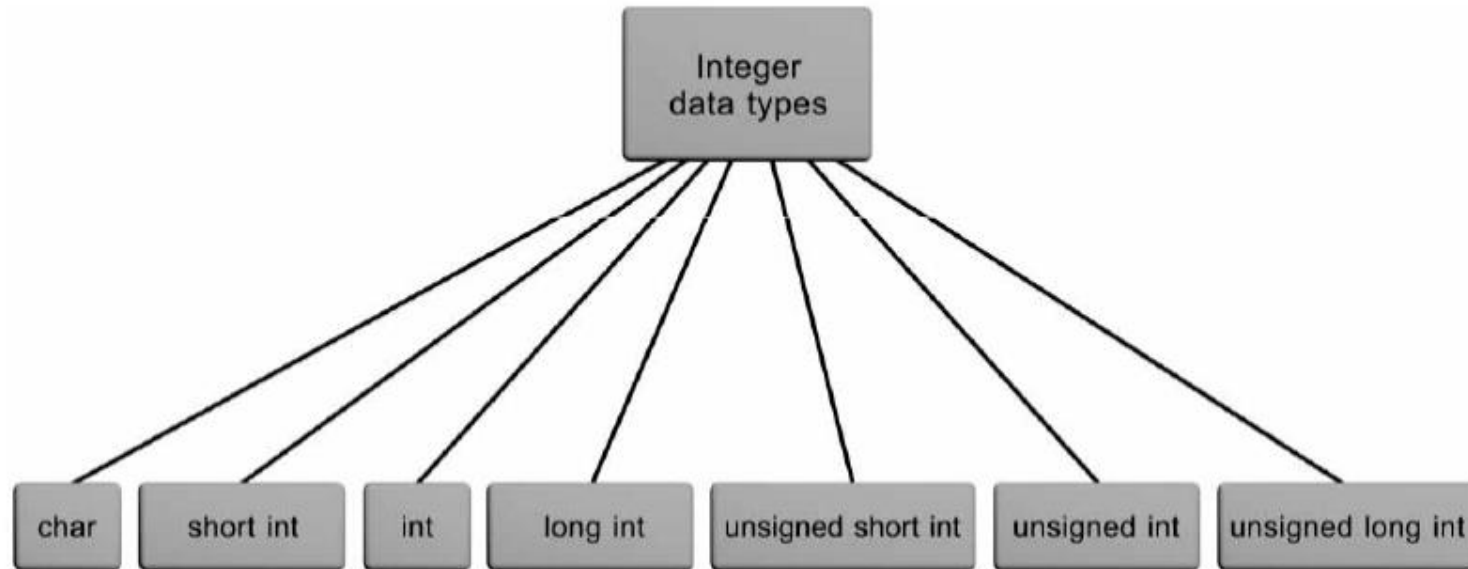
Built-in data types

Data Types (continued)

Data Type	Supplied Operations
Integer	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , <code>=</code> , <code>==</code> , <code>!=</code> , <code><=</code> , <code>>=</code> , <code>sizeof()</code> , and bit operations (see Sec. 14.2)
Floating Point	<code>+</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>=</code> , <code>==</code> , <code>!=</code> , <code><=</code> , <code>>=</code> , <code>sizeof()</code>

C's Built-in Data Types

Integer Data types



C's integer data types

Integer Data types (continued)

- ▶ int: whole numbers (integers)
 - For example: 0, -10, 253, -26351
 - Not allowed: commas, decimal points and special symbols
- ▶ char: stores individual characters (ASCII)
 - For example: 'A', '\$', 'b', '!'

Integer Data Types (continued)

Letter	Code	Letter	Code	Letter	Code	Letter	Code
a	01100001	n	01101110	A	01000001	N	01001110
b	01100010	o	01101111	B	01000010	O	01001111
c	01100011	p	01110000	C	01000011	P	01010000
d	01100100	q	01110001	D	01000100	Q	01010001
e	01100101	r	01110010	E	01000101	R	01010010
f	01100110	s	01110011	F	01000110	S	01010011
g	01100111	t	01110100	G	01000111	T	01010100
h	01101000	u	01110101	H	01001000	U	01010101
i	01101001	v	01110110	I	01001001	V	01010110
j	01101010	w	01110111	J	01001010	W	01010111
k	01101011	x	01111000	K	01001011	X	01011000
l	01101100	y	01111001	L	01001100	Y	01011001
m	01101101	z	01111010	M	01001101	Z	01011010

ASCII and Ansi Letter Codes

Integer Data Types (continued)

Escape Sequence	Character Represented	Meaning	ASCII Code
\n	Newline	Move to a new line	00001010
\t	Horizontal tab	Move to next horizontal tab setting	00001001
\v	Vertical tab	Move to next vertical tab setting	00001011
\b	Backspace	Move back one space	00001000
\r	Carriage return	Carriage return (moves the cursor to the start of the current line—used for overprinting)	00001101
\f	Form feed	Issue a form feed	00001100
\a	Alert	Issue an alert (usually a bell sound)	00000111
\\	Backslash	Insert a backslash character (places an actual backslash character within a string)	01011100
\?	Question mark	Insert a question mark character	00111111
\'	Single quotation	Insert a single quote character (places an inner single quote within a set of outer single quotes)	00100111
\"	Double quotation mark	Insert a double quote character (places an inner double quote within a set of outer double quotes)	00100010
\nnn	Octal number	The number <i>nnn</i> (<i>n</i> is a digit) is to be considered an octal number	—
\xhhhh	Hexadecimal number	The number <i>hhhh</i> (<i>h</i> is a digit) is to be considered a hexadecimal number	—
\0	Null character	Insert the null character, which is defined as having the value 0	00000000

Escape Sequences

Floating-Point data types

- ▶ A floating-point value (real number) can be the number zero or any positive or negative number that contains a decimal point
 - For example: +10.625, 5., -6.2, 3251.92, +2
 - Not allowed: commas, decimal points, special symbols
- ▶ float: single-precision number
- ▶ double: double-precision number
- ▶ float literal is indicated by appending an f or F
- ▶ long double is created by appending an l or L
 - 9.234 indicates a double literal
 - 9.234f float literal
 - 9.234L indicates a long double literal

Floating-Point data types (continued)

Type	Range of Values
float	-1.4012984643e-45 to 3.4028234663e+38
double and long double	-4.9406564584124654e-324 to 1.7976931348623158e+308

Floating-point Data Types

Exponential Notation

- ▶ In numerical theory, the term precision typically refers to numerical accuracy

Decimal Notation	Exponential Notation
1625.	1.625e3
63421.	6.3421e4
.00731	7.31e-3
.000625	6.25e-4

Decimal Numbers Expressed in Exponential Notation

Arithmetic Operations

- ▶ Arithmetic operators: operators used for arithmetic operations:
 - Addition +
 - Subtraction -
 - Multiplication *
 - Division /
 - Modulus Division %
- ▶ Binary operators require two operands
- ▶ An operand can be either a literal value or an identifier that has a value associated with it

Arithmetic Operations (continued)

- ▶ A simple binary arithmetic expression consists of a binary arithmetic operator connecting two literal values in the form:

literalValue operator literalValue

- $3 + 7$
 - $12.62 - 9.8$
 - $.08 * 12.2$
 - $12.6 / 2.$
- ▶ Spaces around arithmetic operators are inserted for clarity and can be omitted without affecting the value of the expression

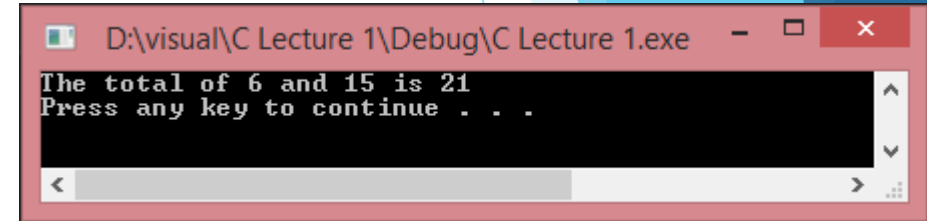
Displaying Numerical Values

Arguments are separated with commas

```
#include <stdio.h>

int main()
{
    printf("The total of 6 and 15 is %d\n", 6 + 15);

    system("pause");
    return 0;
}
```



- ▶ First argument of printf() must be a string
- ▶ A string that includes a conversion control sequence, such as %d, is termed a control string sequence
 - Conversion control sequences are also called conversion specifications and format specifiers
- ▶ printf() replaces a format specifier in its control string with the value of the next argument.
 - In this case, 21

Displaying Numerical Values (continued)

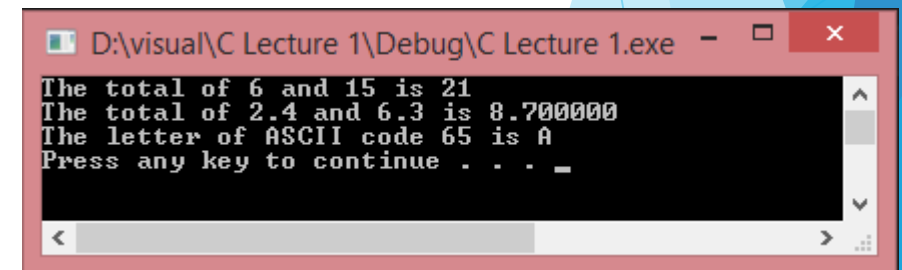
Conversion Control Sequences

Sequence	Meaning
%d	Display an integer as a decimal (base 10) number
%c	Display a character
%f	Display the floating-point number as a decimal number with six digits after the decimal point (pad with zeros, if necessary)

```
#include <stdio.h>

int main()
{
    printf("The total of 6 and 15 is %d\n", 6 + 15);
    printf("The total of 2.4 and 6.3 is %f\n", 2.4 + 6.3);
    printf("The letter of ASCII code %d is %c\n", 65, 65);

    system("pause");
    return 0;
}
```



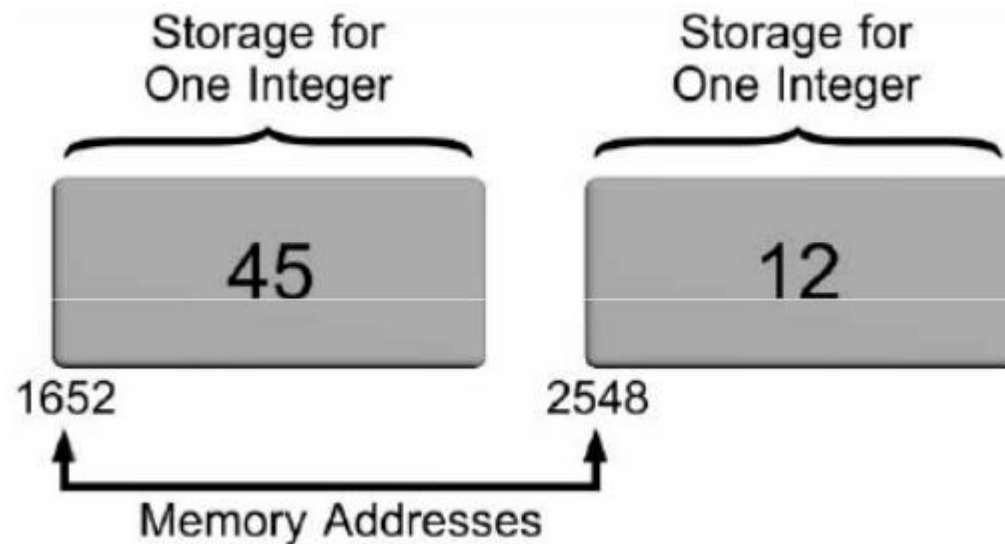
The screenshot shows a Windows command prompt window titled "D:\visual\C Lecture 1\Debug\C Lecture 1.exe". The output of the program is displayed as follows:

```
The total of 6 and 15 is 21
The total of 2.4 and 6.3 is 8.700000
The letter of ASCII code 65 is A
Press any key to continue . . . _
```

Check Point #2

Variables and Declarations

- ▶ Variables are names given by programmers to computer storage
- ▶ Variable name usually limited to 255 characters
- ▶ Variable names are case sensitive

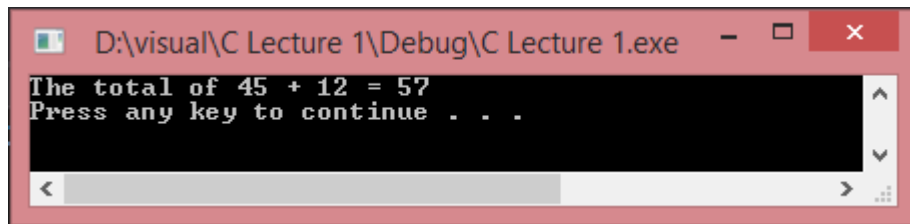
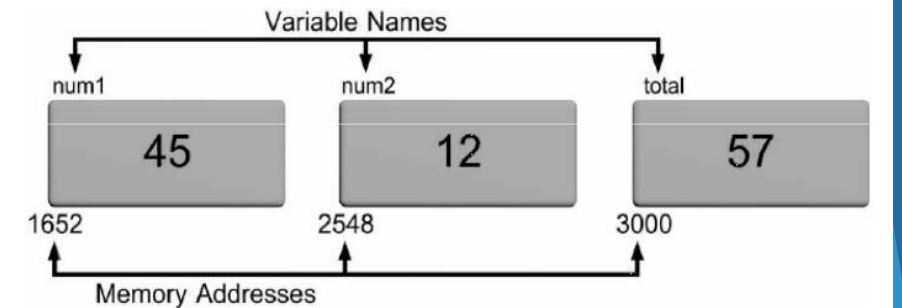


Variables and Declarations (continued)

```
#include <stdio.h>

int main()
{
    int num1 = 45;
    int num2 = 12;
    int total = num1 + num2;
    printf("The total of %d + %d = %d\n", num1, num2, num1 + num2);

    system("pause");
    return 0;
}
```



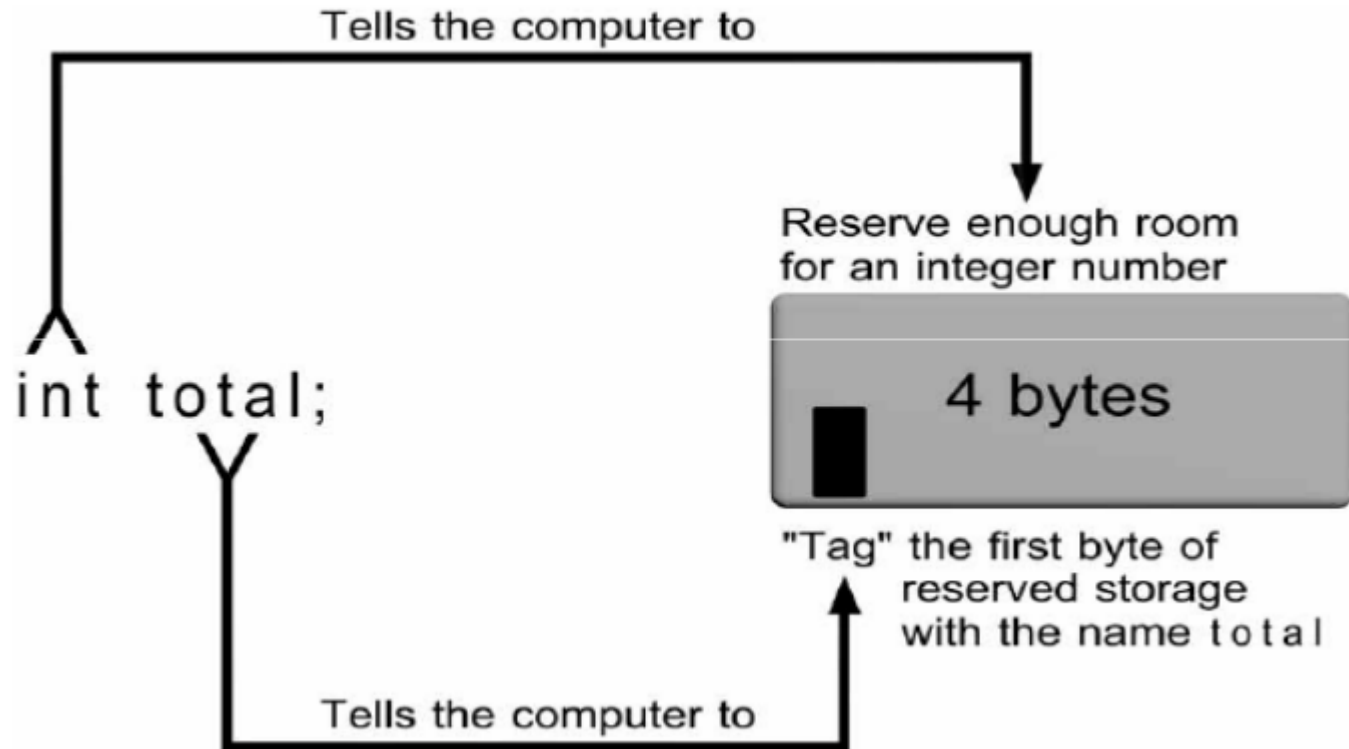
Declaration Statements

- ▶ Naming and specifying the data type that can be stored in each variable is accomplished using declaration statements
- ▶ Declaration statements within a function appear immediately after the opening brace of a function

```
function name()  
{  
    declaration statements;  
    other statements;  
}
```

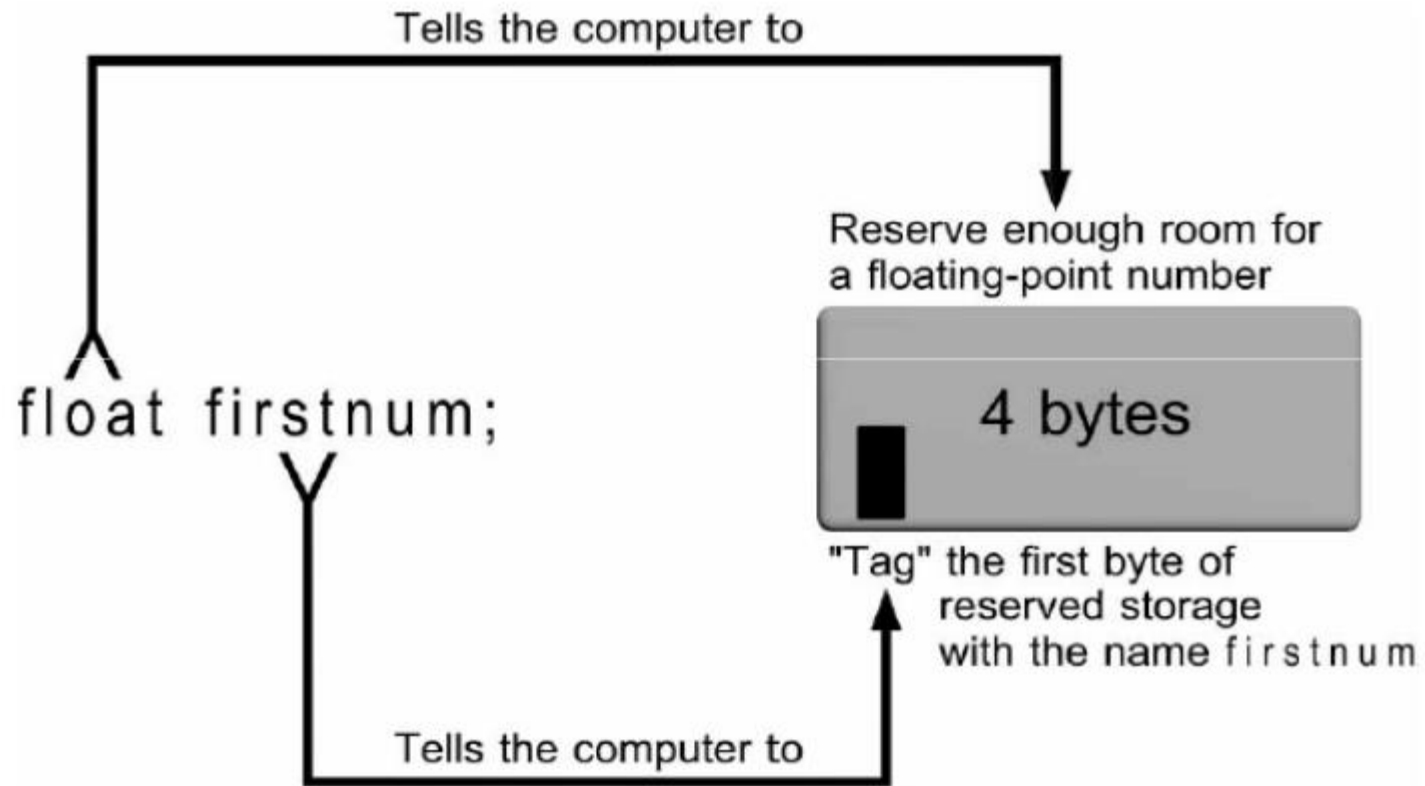
- ▶ Definition statements define or tell the compiler how much memory is needed for data storage

Declaration Statements (continued)



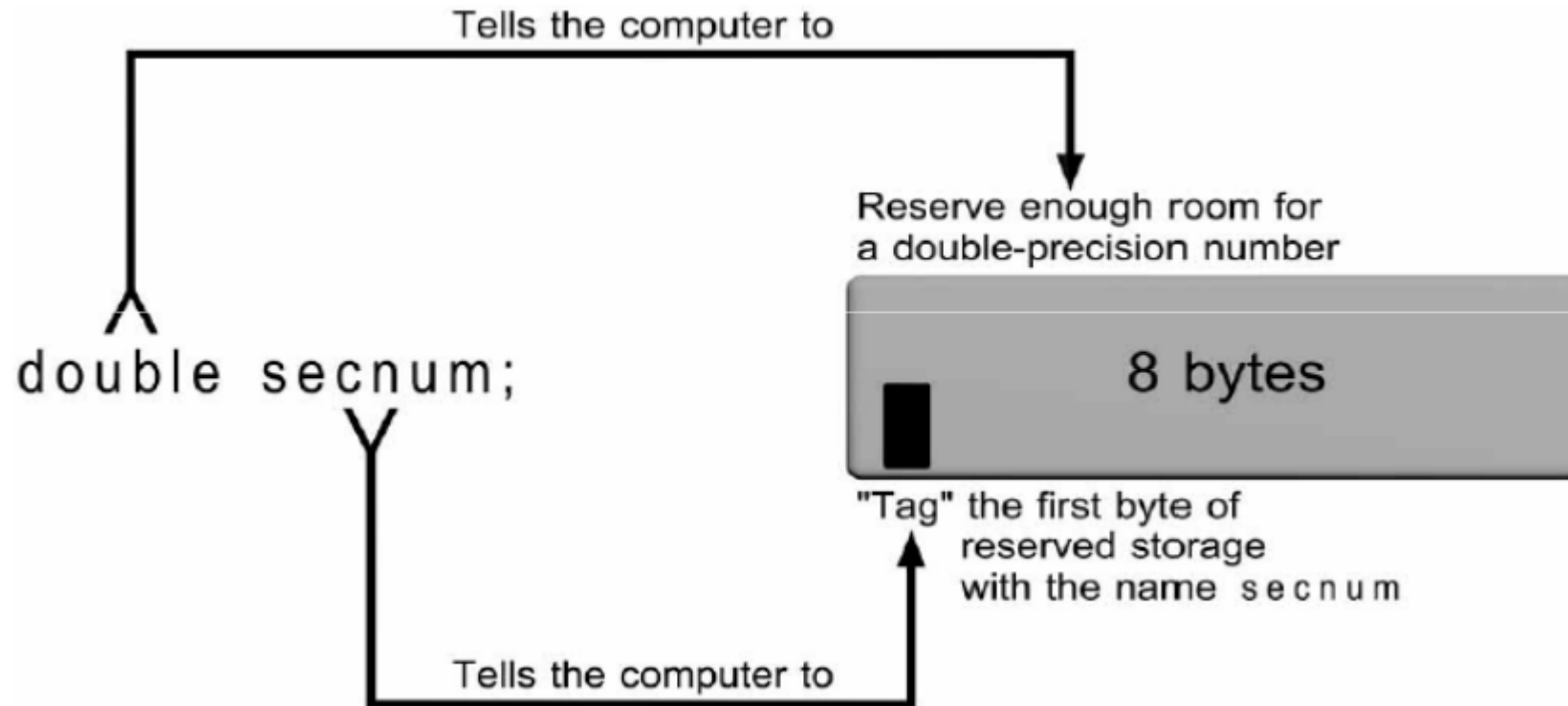
Defining the integer variable named total

Declaration Statements (continued)



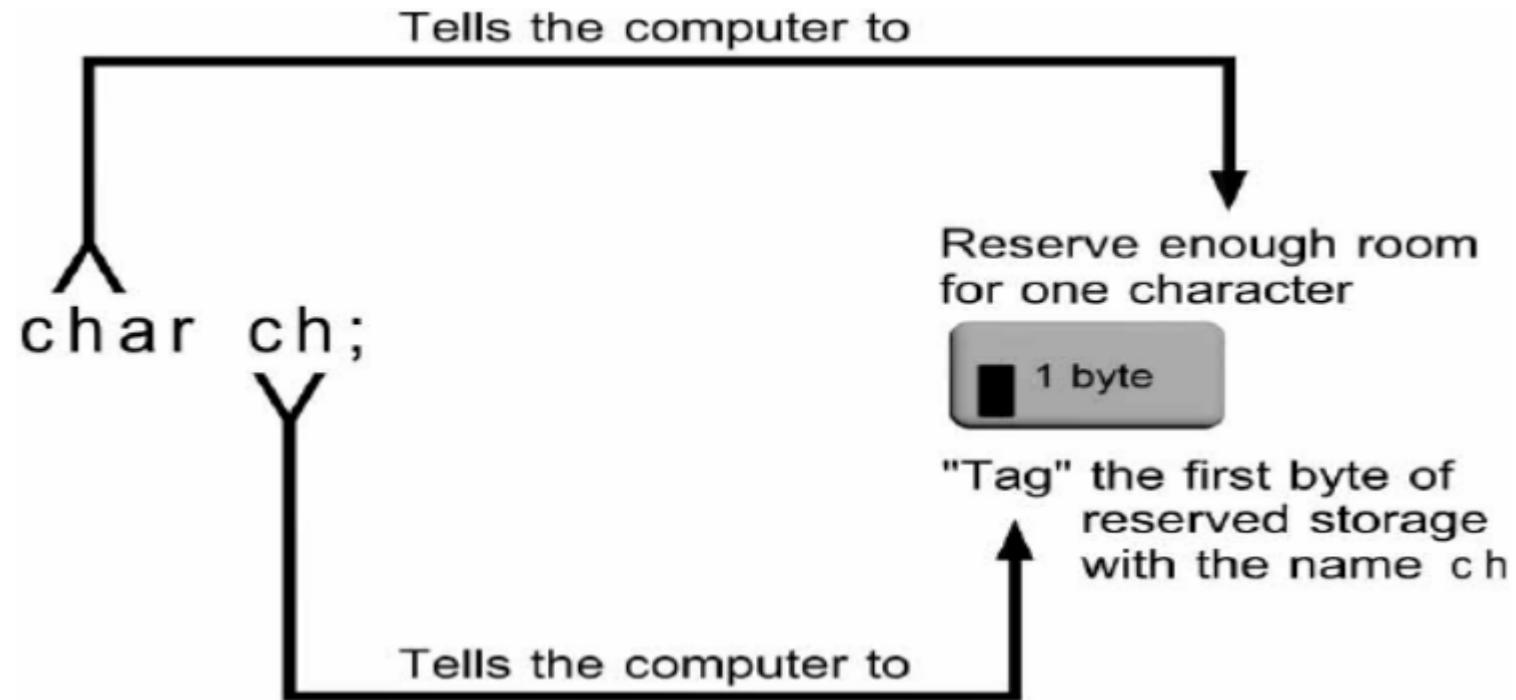
Defining the floating-point variable named `firstnum`

Declaration Statements (continued)



Defining the double-precision variable named `secnum`

Declaration Statements (continued)



Defining the character variable named ch

Selecting Variable Names

- ▶ Make variable names descriptive
- ▶ Limit variable names to approximately 20 characters
- ▶ Start the variable name with a letter, rather than underscore (_)
- ▶ In a variable name consisting of several words, capitalize the first letter of each word after the first
- ▶ Use variable names that indicate what the variable corresponds to, rather than how it is computed
- ▶ Add qualifiers, such as Avg, Min, Max, and Sum to complete a variable's name where appropriate
- ▶ Use single-letter variable names, such as i, j, and k, for loop indexes

Initialization

- ▶ Declaration statements can be used to store an initial value into declared variables
- ▶ When a declaration statement provides an initial value, the variable is said to be initialized
- ▶ Literals, expressions using only literals such as $87.0 + 12 - 2$, and expressions using literals and previously initialized variables can all be used as initializers within a declaration statement

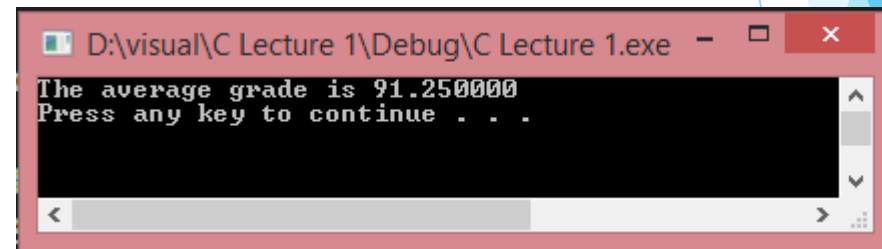
Case Study

```
#include <stdio.h>

int main()
{
    float grade1, grade2, total, average;

    grade1 = 85.5;
    grade2 = 97.0;
    total = grade1 + grade2;
    average = total / 2.0;
    printf("The average grade is %f\n", average);

    system("pause");
    return 0;
}
```



Check Point #3

References

- ▶ A First Book of ANSI C, Fourth Edition