Introduction C

Lecture 9

C PROGRAMMING

Outline

This lecture covers:

- Variables
- Data Types
- Functions; Printf, Scanf, .. Etc.
- Tracing a Program
- Memory Concepts
- C operators (Arithmetic, equality & relational)
- Operators Precedence

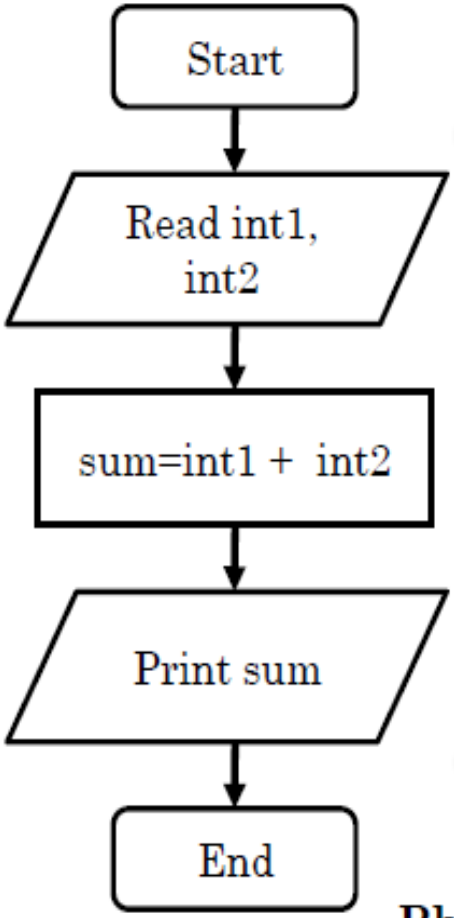
PROBLEM SOLVING STEPS

- Define the problem
- Analyze the problem
- Develop an algorithm (a method) for solving a problem
- Write a computer program corresponding to the algorithm
- Test and debug the program
- Document the program. (write an explanation of how the program works and how to use it.)

ADDING 2 INTEGERS

A Variable name: a place in the memory where you store a Value of a certain Type.

Problem : Adding of two integer numbers.



=

```
#include <stdio.h>

int main( void )
{
  int int1; // Number 1 to enter
  int int2; //Number 2 to enter
  int sum;   //in which the summation stored

  scanf ("%d", & int1); // Read Num1
  scanf ("%d", & int2); // Read Num2
  sum = int1 + int2;   // Sum Num1 and Num2
  printf (" Sum is %d\n", sum); // Print Sum

   return 0;
}
```

Variables declarations

Phase 1 Phase 2

VARIABLE DECLARATION

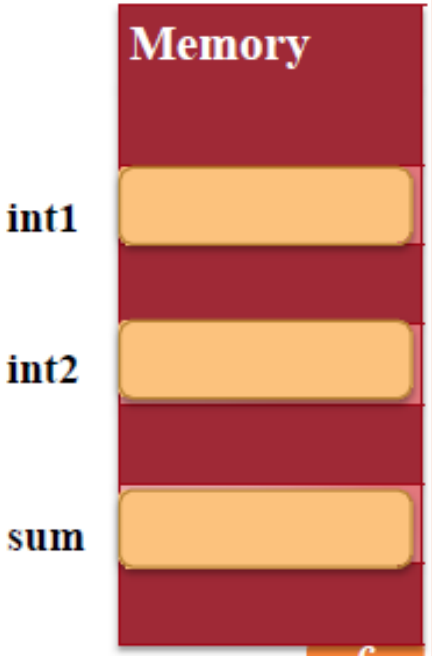
```
int int1;
int int2;
int sum;
scanf ("%d", & int1);
scanf ("%d", & int2);
sum = int1 + int2;
printf ("sum   is %d\n",sum);
```

- Lines

```
int  int1;  // Number 1 to enter
int  int2;  // Number 2 to enter
int  sum;   // Summation
```

- Are definitions for variables.
- Could be combined (only when having the same type) into:

```
int int1, int2, sum;
```



A Variable

```
int int1;
int int2;
int sum;
scanf ("%d", & int1);
scanf ("%d", & int2);
sum = int1 + int2;
printf ("Sum is %d\n",sum);
```

- Lines

```
int int1;  // Number1
int int2;  // Number2
int sum;   // Summation
```

Type = Data Types

Name = Identifier

Declaring a variable means specifying both its identifier and its data type

Identifier Rules

- <u>Rules of identifier:</u>

1) Consists of: A-Z, a-z, 0-9, and (_)
2) Case sensitive : a1 ≠ A1
3) Should Not be a keyword
4) Should start with a letter

underscore

Good Practices:
1) Meaningful identifiers (Ex: sideLength**)**
2) <= 31 characters
3) For single syllables, use small letters
4) For multiple syllables, use (_) or begin each syllable with a capital letter. Ex: total_cost, TotalCost

Example of Identifiers

- VALID

  age_of_person taxRateY2K
  PrintHeading ageOfHorse

- NOT VALID (Why)?

  age# 2008TaxRate Age-Of-Cat

Standard Data Types in C

- **Integral Types**
  - represent whole numbers and their negatives
  - declared as int, short, or long
- **Floating Types**
  - represent real numbers with a decimal point
  - declared as float, or double
- **Character Types**
  - represent single characters
  - declared as char

Note:
Each data type reserve different number of bytes
Ex: char (1 byte) and
int (2 bytes)

Samples of Data Values

int sample values

4578 **- 4578** **0**

float sample values

95.274 **95.0** **0.265**

char sample values

'B' 'd' '4' '?' '*'

Variable Assignment: Giving a Value to a Variable

You can assign (give) a value to a variable by using the assignment operator =

Example:

```
middleInitial = 'R' ;

age = 12;

tax = 3.7;

x = ( 7- 10 )* (5 % 3) * 4 + 9
```

What is the Content of Variables A and B After Each Step?

Trace this code segment:

```
int a , b ;
a = 5 ;
b = 7 ;
b = a +10  ;
a = b - 1 ;
b = 3 ;
printf ( "a= %d, b= %d", a, b);
```

In memory	
a	**b**
5	7
14	15
	3

Output Screen
a = 14 , b= 3

Note: the assignment statement overwrites old values with new ones (destructive behavior)

NOTICE:

```
int  int1;
int  int2;
int sum;
scanf ("%d", & int1);
scanf ("%d", & int2);
sum = int1 + int2;
printf (" Sum is %d\n", sum);
```

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

Fig. 2.5 | Addition program. (Part 1 of 2.)

```
int int1;
int int2;
int sum;
scanf ("%d", & int1);
scanf ("%d", & int2);
sum = int1 + int2;
printf (" Sum is %d\n", sum);
```

THE SCANF FUNCTION

- Line

```
scanf( "%d", &int1 ); // read an integer
```

- function reads from the standard input (keyboard).

format control string

Ampersand (&)
→ address operator

Must exist

Variable name (int1)

Conversion Specifier	Used for data type
%d	int
%f	float
%f	double
%c	char

&int1, tells scanf the location (or address) in memory of int1 then stores the value at that location.

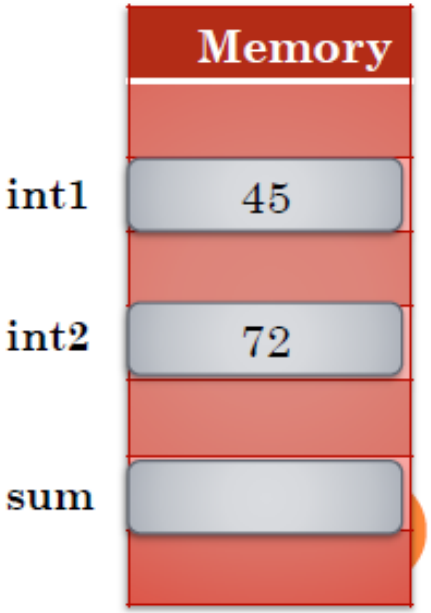
scanf Syntax?

Tracing program

```
int int1;
int int2;
int sum;
scanf ("%d", & int1);
scanf ("%d", & int2);
sum = int1 + int2;
printf (" Sum is %d\n", sum);
```

- Trace is doing what program will do
- Line

```
scanf( "%d", &int1 ); // read an integer1
scanf( "%d", &int2 ); // read an integer2
```



- Could be combined:

```
scanf( "%d %d ", &int1,&int2 );
```

Prompting Messages

```
int int1;
int int2;
int sum;
scanf ("%d", & int1);
scanf ("%d", & int2);
sum = int1 + int2;
printf ("sum is %d\n", sum);
```

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully
} /* end function main */
```

Should be used

Prompt
Tells the user what to do

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

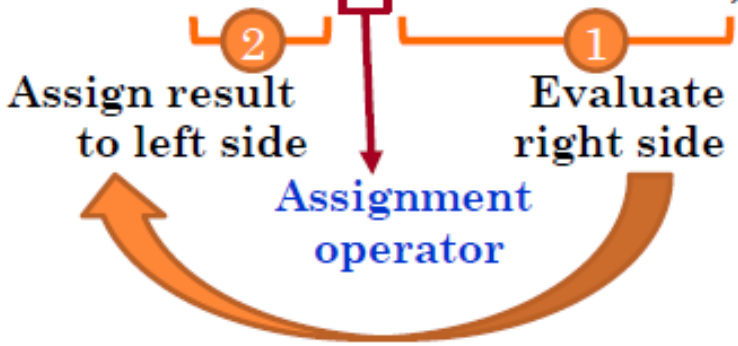
Fig. 2.5 | Addition program. (Part 1 of 2.)

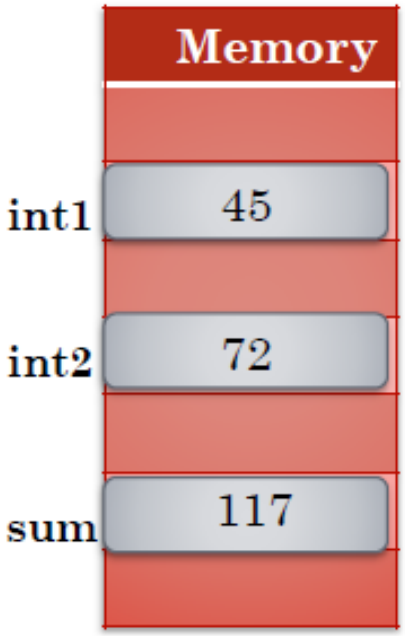
Assignment statement

```
int int1;
int int2;
int sum;
scanf ("%d", & int1);
scanf ("%d", & int2);
sum = int1 + int2;
printf (" Sum is %d\n", sum);
```

- The assignment statement is in line

```
sum = int1 + int2; // assign total to sum
```





Screen

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

Any changes on screen?

The printf with format control string

```
int int1;
int int2;
int sum;
scanf ("%d", & int1);
scanf ("%d", & int2);
sum = int1 + int2;
printf (" Sum is %d\n", sum);
```

- Line

```
printf( "Sum is %d\n", sum ); //print sum
```

Print integer value (%d) here

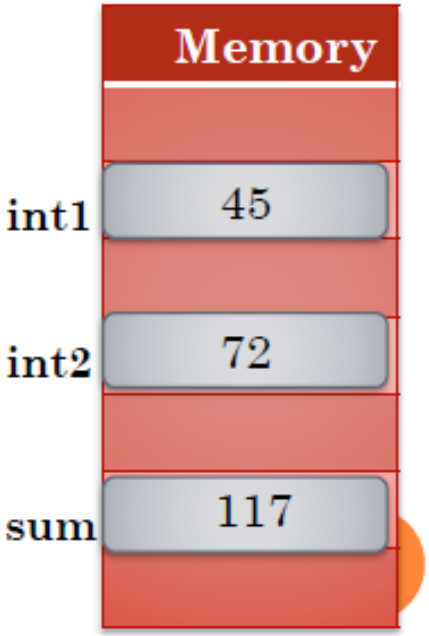
The value to be printed

Conversion specifier depend on variable type

More general printf Syntax?

Screen

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



Calculations in printf statements

```
int int1;
int int2;
int sum;
scanf ("%d", & int1);
scanf ("%d", & int2);
sum = int1 + int2;
printf (" Sum is %d\n", sum);
```

- These 2 Lines:

```
sum = int1 + int2; // assign total to sum
printf( "Sum is %d\n", sum ); //print sum
```

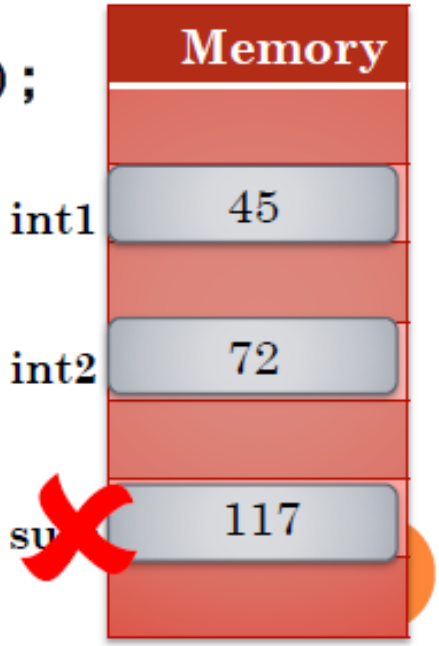
- Could combined into

```
printf( "Sum is %d\n", int1 + int2 );
```

Is there differences?

Screen

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



CERT Center

7th edition

- The CERT (Computer Emergency Response Team) was created to analyze and respond promptly to attacks

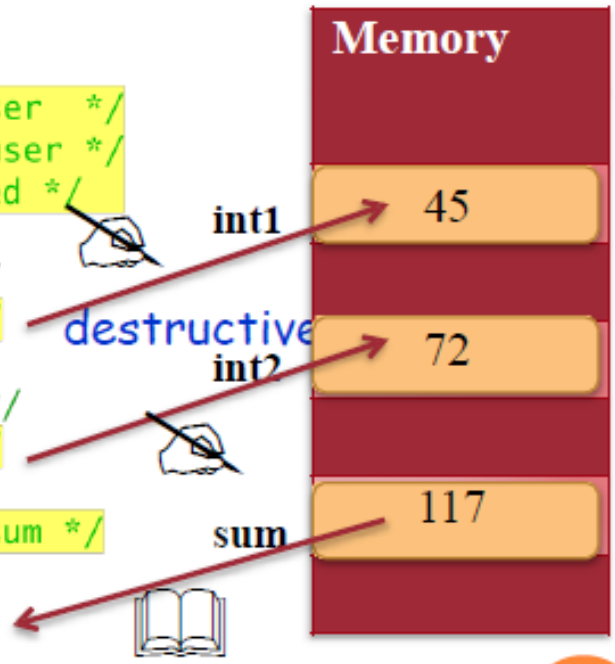
- CERT recommendations <u>to avoid programming practices that open systems to attacks:</u>
  - ***Avoid single-argument printf***
    Ex:
    printf("Welcome to C!\n"); ➔ puts ("Welcome to C!");
    printf("Welcome "); ➔ printf("%s", "Welcome ");

  - ***Use scanf_s and printf_s***

Memory Concepts

Sum = integer1 + integer2
Destructive or non-destructive?

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully
} /* end function main */
```



Non-destructive

Fig. 2.5 | Addition program. (Part 1 of 2.)

Addition Program

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully */
} /* end function main */
```


Fig. 2.5 | Addition program. (Part 1 of 2.)

Arithmetic Operators

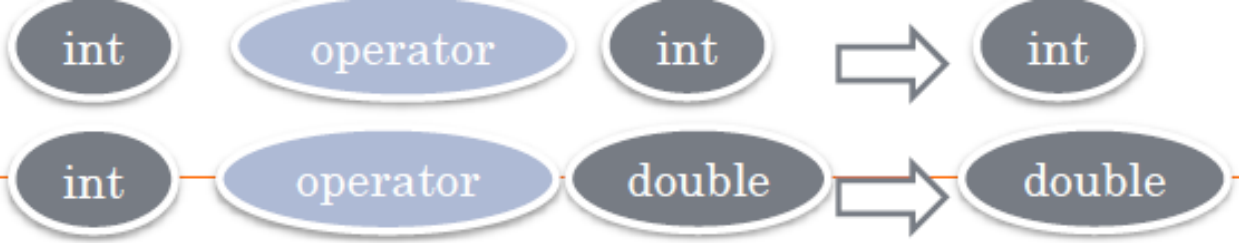
C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	f + 7	f + 7
Subtraction	-	p - c	p – c
Multiplication	*	bm	b * m
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	x / y
Remainder	%	r mod s	r % s

modulus

- Asterisk (*) indicates multiplication
- Percent sign (%) denotes remainder operator

- Arithmetic operators are all binary operators; each has 2 operands
  - Ex: The expression 3 + 7 contains:
    - the binary operator (+) and the operands 3 and 7

Division



- Integer division (both operands are integers)

➔ result is an integer value

Ex:

- 6/4 = 1 and 18/5 = 3

- If one of the operands is not an integer number

➔ result is not an integer value

Ex:

- 18.0/5= 3.6

- **Take Care: Division by zero**

➔ It results in a fatal error: program terminates immediately

Mixed type expression Example

Remember
The expression is evaluated before the assignment is made

```
int m, n;
double p, x, y;
m = 3 ; n = 2 ; p = 2.0 ;
x = m / p ;
y = m / n ;
m= m / p;
```

m	n	p	x	y
3	2	2.0	1.5	1.0
1				

Remainder Operator (%)

- It is an integer operator
  → only integer operands

- It gets the remainder of the division
  Ex:
  - 7 % 4 = 3
  - 17 % 5 = 2

- The magnitude of **m%n** must always be less than the divisor **n**

Ex:
The result of 299 % 100 is 99
→ the result must be between 0 and 99.

Precedence of Arithmetic Operators

High Precedence

low Precedence

Operator	Operation	Order of evaluation (precedence)
( )	Parentheses	From inner to outer, from left to right
* / %	Multiplication Division Remainder	From left to right
+ -	Addition Subtraction	From left to right

Sample Algebraic and C Expressions

Algebra:

$$m = \frac{a + b + c + d + e}{5}$$

$$y = mx + b$$

$$z = pr\%q + w/x - y$$

C: straight-line form

$m = (a + b + c + d + e)/5$ ☑ (1)

$m = a + b + c + d + e/5$ ☒ (1)

```
y = m * x + b;
```

(1)

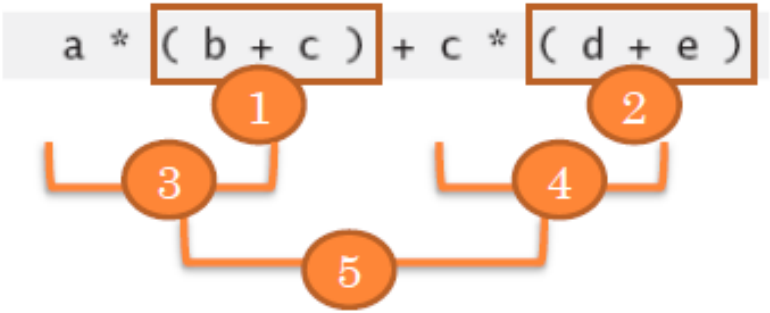
```
z = p * r % q + w / x - y;
```

(1) (2) (3) (4) (5) (6)

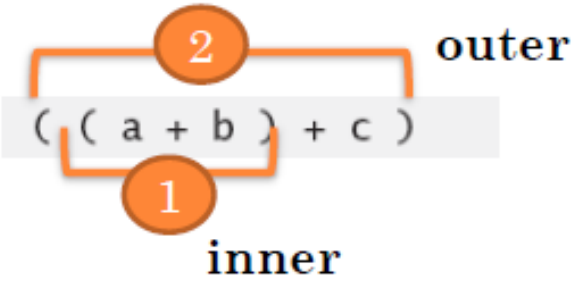
29

Expressions evaluation

Parentheses on the same level



Nested parentheses



Redundant parentheses?

unnecessary

Evaluate the Expression

7 * 10 - 5 % 3 * 4 + 9

7 * 10 - 5 % 3 * 4 + 9

70 - 5 % 3 * 4 + 9

70 - 5 % 3 * 4 + 9

70 - 2 * 4 + 9

70 - 2 * 4 + 9

70 - 8 + 9

70 - 8 + 9

62 + 9

71

PARENTHESES (BRACKETS)

- parentheses can be used to change the usual order
- parts in ( ) are evaluated first
- evaluate **(7 * (10 - 5) % 3) * 4 + 9**

(7 * 5 % 3) * 4 + 9

(35 % 3) * 4 + 9

2 * 4 + 9

8 + 9

17