CENG 230

Introduction to C Programming

Week 3 – Overview of C

Sinan Kalkan

Some slides/content are borrowed from Tansel Dokeroglu, Nihan Kesim Cicekli.

Notes

Previously on CEng 230!

- The lectures notes:

  http://www.kovan.ceng.metu.edu.tr/~sinan/ceng230/

  - You can also just google my name → Follow “Courses” → “Ceng 230”

- Location:
  - Starting from “5 March, 2015”, lectures will be held in BMB-1 (in Computer Engineering Dept.)

- Midterm:
  - 28 April, 2015 at 17:40

Previously on CEng 230!

Variables and identifiers

Previously on CEng 230!

C has the following basic built-in datatypes.

- int
- float
- double
- char

TABLE 2.4 Type double Constants (real numbers)

Valid double Constants	Invalid double Constants
`3.14159`	`150` (no decimal point)
`0.005`	`.12345e` (missing exponent)
`12345.0`	`15e-0.3` (`0.3` is invalid exponent)
`15.0e-04` (value is `0.0015`)	
`2.345e2` (value is `234.5`)	`12.5e.3` (`.3` is invalid exponent)
`1.15e-3` (value is `0.00115`)	`34,500.99` (comma is not allowed)
`12e+5` (value is `1200000.0`)	

Previously on CEng 230!

Valid Identifiers

```
letter_1, letter_2, inches, cent, CENT_PER_INCH, Hello, variable
```

TABLE 2.2 Invalid Identifiers

Invalid Identifier	Reason Invalid
`1Letter`	begins with a letter
`double`	reserved word
`int`	reserved word
`TWO*FOUR`	character * not allowed
`joe's`	character ' not allowed

int1 and Int1 are not the same identifiers/variables

Previously on CEng 230!

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

Keywords added in C99

`_Bool` `_Complex` `_Imaginary` `inline` `restrict`

Fig. 2.15 | C's keywords.

Previously on CEng 230!

Basic Input/Output in C

Output

Previously on CENG 230!

- **printf**(format string, var1, var2, …)
  - Format string contains:
    - d,i: integers
    - f: float, double
    - e: float, double in exponential notation
    - c: character
    - s: string

Input

Previously on CENG 230!

- **scanf**(format string, &var1, &var2, …)
  - var1, var2, ..: variables!
  - Format string contains:
    - d,i: integers
    - f: float, double
    - e: float, double in exponential notation
    - c: character
    - s: string

Previously on CEng 230!

```
1  /* Fig. 2.3: fig02_03.c
2     Printing on one line with two printf statements */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome " );
9     printf( "to C!\n" );
10
11    return 0; /* indicate that program ended successfully */
12 } /* end function main */
```

Fig. 2.3 | Printing on one line with two `printf` statements. (Part 1 of 2.)

```
Welcome to C!
```

Previously on CEng 230!

```
1  /* Fig. 2.4: fig02_04.c
2     Printing multiple lines with a single printf */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome\nto\nC!\n" );
9
10    return 0; /* indicate that program ended successfully */
11 } /* end function main */
```

```
Welcome
to
C!
```

Fig. 2.4 | Printing multiple lines with a single `printf`.

Previously on CEng 230!

Escape sequence	Description
\n	Newline. Position the cursor at the beginning of the next line.
\t	Horizontal tab. Move the cursor to the next tab stop.
\a	Alert. Sound the system bell.
\\	Backslash. Insert a backslash character in a string.
\"	Double quote. Insert a double-quote character in a string.

Fig. 2.2 | Some common escape sequences .

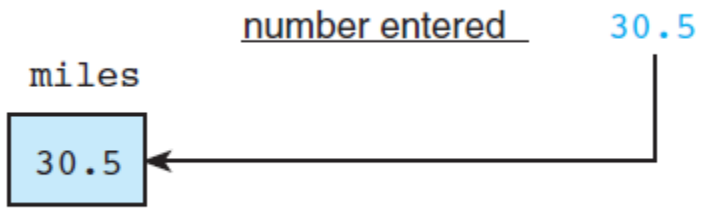
Previously on CEng 230:

TABLE 2.8 Placeholders in Format Strings

Placeholder	Variable Type	Function Use
%c	char	printf/scanf
%d	int	printf/scanf
%f	double	printf
%lf	double	scanf

FIGURE 2.6

Effect of `scanf("%lf", &miles);`




```
int first, second;
scanf("%d%d", &first, &second);
```

```
double miles;  /* distance in miles        */
scanf("%lf", &miles);
```

Previously on CEng 230!

```
1  /* Fig. 2.5: fig02_05.c
2     Addition program */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     int integer1; /* first number to be input by user  */
9     int integer2; /* second number to be input by user */
10    int sum; /* variable in which sum will be stored */
11
12    printf( "Enter first integer\n" ); /* prompt */
13    scanf( "%d", &integer1 ); /* read an integer */
14
15    printf( "Enter second integer\n" ); /* prompt */
16    scanf( "%d", &integer2 ); /* read an integer */
17
18    sum = integer1 + integer2; /* assign total to sum */
19
20    printf( "Sum is %d\n", sum ); /* print sum */
21
22    return 0; /* indicate that program ended successfully */
23 } /* end function main */
```

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

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

Formating the output of integer values

Previously on CEng 230!

Specifying the format of an integer value displayed by a C program is fairly easy. You simply add a number between the `%` and the `d` of the `%d` placeholder in the `printf` format string. This number specifies the **field width**—the number of columns to use for the display of the value. The statement

```
printf("Results: %3d meters = %4d ft. %2d in.\n",
       meters, feet, inches);
```

indicates that `3` columns will be used to display the value of `meters`, `4` columns will be used for `feet`, and `2` columns will be used for `inches` (a number between `0` and `11`). If `meters` is `21`, `feet` is `68`, and `inches` is `11`, the program output will be

```
Results:    21 meters =    68 ft. 11 in.
```

TABLE 2.14 Displaying 234 and –234 Using Different Placeholders

Value	Format	Displayed Output	Value	Format	Displayed Output
234	%4d	▮234	-234	%4d	-234
234	%5d	▮▮234	-234	%5d	▮-234
234	%6d	▮▮▮234	-234	%6d	▮▮-234
234	%1d	234	-234	%2d	-234

Formating the output of double values

Previously on CEng 2301

TABLE 2.16 Formatting Type double Values

Value	Format	Displayed Output	Value	Format	Displayed Output
3.14159	%5.2f	▮3.14	3.14159	%4.2f	3.14
3.14159	%3.2f	3.14	3.14159	%5.1f	▮▮3.1
3.14159	%5.3f	3.142	3.14159	%8.5f	▮3.14159
.1234	%4.2f	0.12	-.006	%4.2f	-0.01
-.006	%8.3f	▮▮-0.006	-.006	%8.5f	-0.00600
-.006	%.3f	-0.006	-3.14159	%.4f	-3.1416

Previously on CEng 230!

Operators and Expressions

Previously on CEng 230!

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	%	$r \bmod s$	`r % s`

printf("Welcome to \%d", (3/2));

Output is : 1

Previously on CEng 230!

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
*	Multiplication	Evaluated second. If there are several, they're evaluated left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated last. If there are several, they're evaluated left to right.
-	Subtraction	

Previously on CEng 230!

TABLE 2.9 Arithmetic Operators

Arithmetic Operator	Meaning	Examples
+	addition	`5 + 2 is 7` `5.0 + 2.0 is 7.0`
-	subtraction	`5 - 2 is 3` `5.0 - 2.0 is 3.0`
*	multiplication	`5 * 2 is 10` `5.0 * 2.0 is 10.0`
/	division	`5.0 / 2.0 is 2.5` `5 / 2 is 2`
%	remainder	`5 % 2 is 1`

Previously on CEng 230!

TABLE 2.10 Results of Integer Division

`3 / 15 = 0`	`18 / 3 = 6`
`15 / 3 = 5`	`16 / -3 =` -5
`16 / 3 = 5`	`0 / 4 = 0`
`17 / 3 = 5`	`4 / 0` is undefined

TABLE 2.11 Results of % Operation

`3 % 5 = 3`	`5 % 3 = 2`
`4 % 5 = 4`	`5 % 4 = 1`
`5 % 5 = 0`	`15 % 5 = 0`
`6 % 5 = 1`	`15 % 6 = 3`
`7 % 5 = 2`	`15 % -7` = 1
`8 % 5 = 3`	`15 % 0` is undefined

Previously on CEng 230!

TABLE 2.13 Mathematical Formulas as C Expressions

Mathematical Formula	C Expression
1. $b^2 - 4ac$	`b * b - 4 * a * c`
2. $a + b - c$	`a + b - c`
3. $\frac{a + b}{c + d}$	`(a + b) / (c + d)`
4. $\frac{1}{1 + x^2}$	`1 / (1 + x * x)`
5. $a \times -(b + c)$	`a * -(b + c)`

Previously on CEng 230!

Rules for Evaluating Expressions

a. *Parentheses rule:* All expressions in parentheses must be evaluated separately. Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.

b. *Operator precedence rule:* Operators in the same expression are evaluated in the following order:

unary +, -	first
*, /, %	next
binary +, -	last

c. *Associativity rule:* Unary operators in the same subexpression and at the same precedence level (such as + and -) are evaluated right to left (*right associativity*). Binary operators in the same subexpression and at the same precedence level (such as + and -) are evaluated left to right (*left associativity*).

Previously on CEng 230!

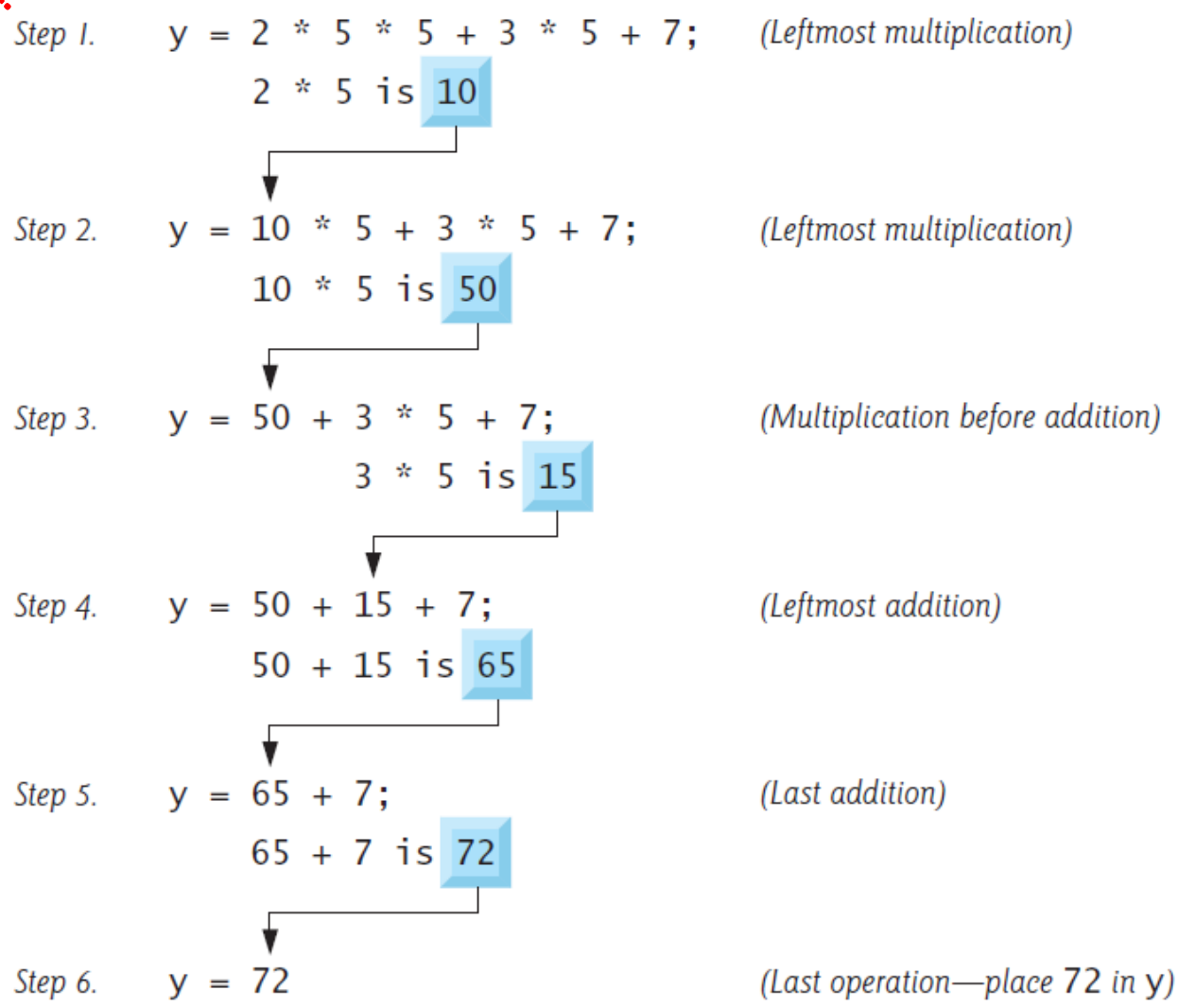


Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

Evaluation of a Second-Degree Polynomial

To develop a better understanding of the rules of operator precedence, let's see how C evaluates a second-degree polynomial.

```
y = a * x * x + b * x + c;
    6   1   2   4   3   5
```

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

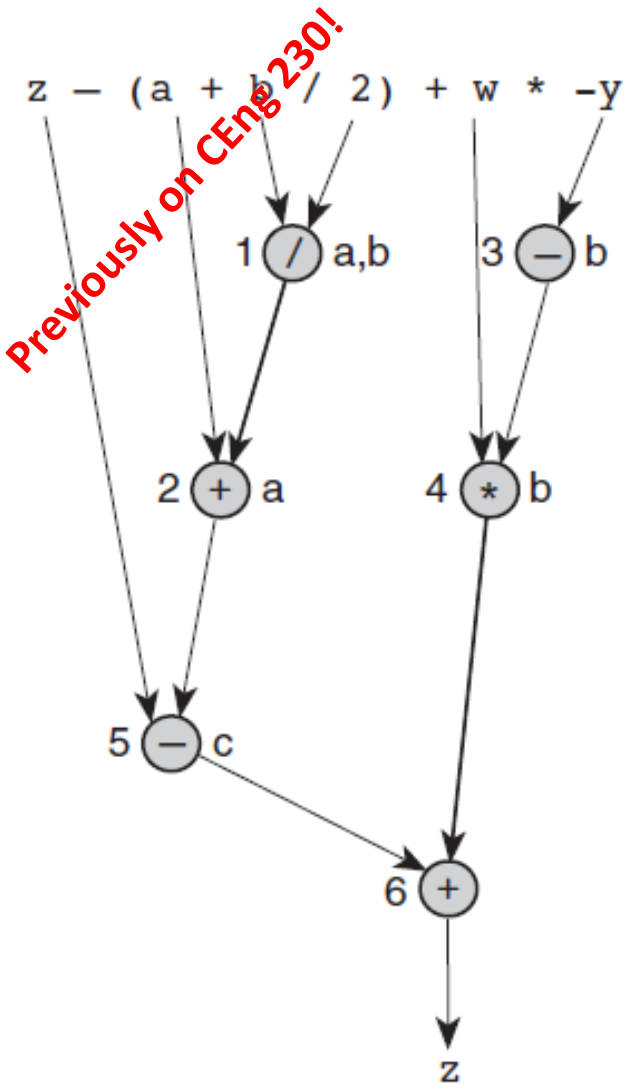
`50 + 15 is 65`

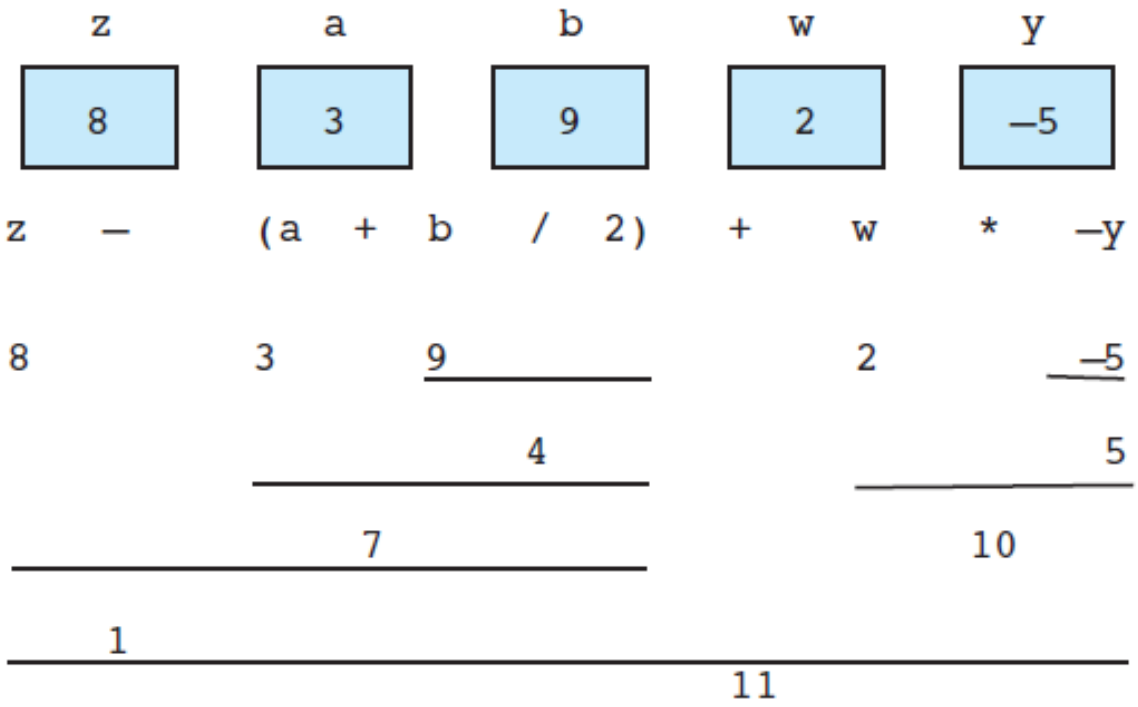
Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72` *(Last operation—place 72 in y)*

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.


z – (a + b / 2) + w * –y
Previously on CEng 230!
1 / a,b
2 + a
3 – b
4 * b
5 – c
6 +
z


z a b w y
8 3 9 2 –5
z – (a + b / 2) + w * –y
8 3 9 2 –5
4 5
7 10
1
11

Today

- Continue with operators
  - Relational operators
  - Increment/decrement operators
  - Logical operators
- Type conversion
- Examples

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.12 | Equality and relational operators.

! exclamation mark

= is assignment and == is an equality operator

Increment, Decrement Operators

Operator	Sample expression	Explanation
++	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Fig. 3.12 | Increment and decrement operators

- ++a, --a

  vs

- a++, a--

Assignment operators

=

+=

-=

*=

/=

%=

a+=10; is the same with

a=a + 10;

Compound Assignment Operators

- var op= expr
- += -= *= /= %=

Some examples

- i += j = k;
- i = j += k;

Relational Operators

- < <= > >= == !=
- False means 0 (zero)
- True means anything that is not False (i.e., non-zero)

Operator	Type	Associativity
+ - ++ --	Unary	Right to left
* / %	Binary	Left to right
+ -	Binary	Left to right
< <= > >=	Binary	Left to right
== !=	Binary	Left to right
= *= /= %= += -=	Binary	Right to left

Example: a = b + c <= d + e == c - d

Homework

- Write a C code that solves the following system of linear equations:

$$ax + by = c$$
$$dx + ey = f$$

- Your program should read a, b, c, d, e and f from standard input, and display the values of x and y to the standard output.

- Bring the print-out of the C code to the next lecture.