Fundamental of Programming (C)

Lecturer: Omid Jafarinezhad

Lecture 4

Input and Output

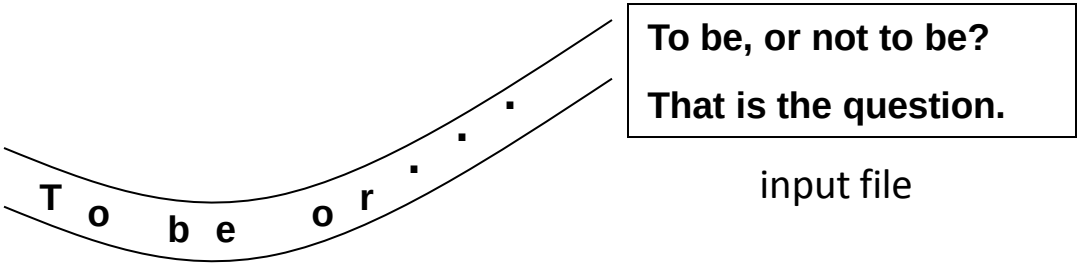
Outline

- printf
- scanf
- putchar
- getchar
- getch
- getche

Input and Output in C

- C has no built-in statements for IO
  - The I/O library functions are in <stdio.h>

- All input and output is performed with streams
  - A stream is a sequence of characters organized into lines



  - Standard types of streams in C:
    - stdin: input stream (normally connected to the keyboard)
    - stdout: output stream (normally connected to the screen)
    - stderr: error stream (normally connected to the screen)

Formatted Output - printf

- This function provides for formatted output to the screen. The syntax is:

  printf ("format", var1, var2, …) ;

- The format includes, some **text** and **conversion specifications**

- A conversion specifier begins with the **%** character. After the **%** character come the following in this order:
  - [**flags**]: control the conversion (**optional**)
  - [**width**]: the number of characters to print (**optional**)
  - [**.precision**]: the amount of precision to print for a number type (**optional**)
  - [**modifier**]: overrides the size (type) of the argument (**optional**)
  - [**type**]: the type of conversion to be applied (**required**)

  - Example: %[**flags**][**width**][**.precision**][**modifier**]**type**

Conversion specifier: type

Type	Output
d, i	Type **signed int**
o	Type **unsigned int** printed in **octal**
u	Type **unsigned int** printed in **decimal**
x	Type **unsigned int** printed in **hexadecimal** as dddd using **a, b, c, d, e, f**
X	Type **unsigned int** printed in **hexadecimal** as dddd using **A, B, C, D, E, F**
f	Type **double** printed as **[-]ddd.ddd**
e, E	Type **double** printed as **[-]d.ddde[-]dd** where there **is one digit printed before the decimal (zero only if the value is zero); the exponent contains at least two digits**. If type is **E** then the exponent is printed with a **capital E**
g, G	Type **double** printed as type **e or E if the exponent is less than -4 or greater than or equal to the precision, otherwise printed as type f**. **Trailing zeros are removed**. **Decimal point character appears only if there is a nonzero decimal digit**
c	Type **char**; **single character** is printed
s	Type **pointer to array**; string is printed according to precision (no precision prints entire string)
p	Prints the **value of a pointer** (the memory address)
n	The argument **must be a pointer to an int**. Stores the **number of characters printed thus far**
%	A **%** sign is printed

Example: type

```
#include <stdio.h>  // note: preprocessor does not need semicolon
/*  Objective: Using conversion specifiers */
int main()
{ // start main
   int i = 1234;
   float m = 12.5, n = -12.5;

   printf("%d %o %x %X\n", i, i, i, i);
   printf("%f %f\n", m, n);
   printf("%e %e %E\n", n, m, n);
   printf("%g %g\n", m, n);
   printf("%s\n", "string");
   printf("-------\n");
   printf("%d %d\n", i);
   printf("%d\n", i, i);
   printf("%d");
   getch();
   return 0; // indicates successful termination
} // end main
```

```
1234 2322 4d2 4D2
12.500000 -12.500000
-1.250000e+001 1.250000e+001 -1.250000E+001
12.5 -12.5
string
-------
1234 1076428800
1234
1234
```

Example: type

```
#include <stdio.h>
/*  Objective: Using conversion specifiers */
int main()
{ // start main
   int i = 1234;
   float m = 12.5, n = -12.5;
   char c = 'A';
   m = c;
   printf("%f\n", 18); //meaningless; if 18 replace by 18.0 then it is OK.
   printf("%f\n", c); //meaningless
   printf("%f\n", m);
   i = c;
   printf("%d\n", i);
   printf("%d\n", c);
   i = m; // i = 65;
   printf("%d\n", i);
   printf("%d\n", m); //meaningless
   printf("%d\n", 12.6); //meaningless
   //printf("%s %s %s", i, m, c); // meaningless OR rumtime-error
   getch();
   return 0; // indicates successful termination
} // end main
```

```
0.000000
0.000000
65.000000
65
65
65
0
858993459
Press any key to continue . . .
```

Modifier

modifier	type	Effect
h	d, i, o, u, x, X	Value is first converted to a short int or unsigned short int.
h	n	Specifies that the pointer points to a short int.
l	d, i, o, u, x, X	Value is first converted to a long int or unsigned long int .
l	n	Specifies that the pointer points to a long int.
L	e, E, f, g, G	Value is first converted to a long double.

Width

- The width of the field is specified here with a decimal value
- If the value is not large enough to fill the width, then the rest of the field is padded with spaces (unless the 0 flag is specified)
- If the value overflows the width of the field, then the field is expanded to fit the value
- If a * is used in place of the width specifer, then the next argument (which must be an int type) specifies the width of the field.
  - printf("%*f", 7, 98.736);

Example

```
int i = 1234;
float m = 12.5;
float n = -12.5;
printf("%d%d%d\n", i, i, i);
printf("%3d  %5d  %8d\n", i, i, i);
printf("%3s  %5s\n", "test", "c");
printf("%3f  %5f  %12f\n", m, n, n);
```

```
1234123412341234
1234  1234      1234
test    c
12.500000  -12.500000   -12.500000
```

```
printf("%3g %6g %8g\n", m, n, n);
```

```
12.5  -12.5    -12.5
```

Precision

- The precision begins with a dot (.) to distinguish itself from the width specifier
- The precision can be given as a decimal value or as an asterisk (*)
  - If a * is used, then the next argument (which is an int type) specifies the precision
  - when using the * with the width and/or precision specifier, the width argument comes first, then the precision argument, then the value to be converted
  - printf("%*.*f", 7, 2, 98.736);
- Precision does not affect the c type

Precision

.precision	Result

(none)	• Default precision values:
	  • 1 for d, i, o, u, x, X types. The minimum number of digits to appear
	  • 6 for f, e, E types. Specifies the number of digits after the decimal point
	• For g or G types all significant digits are shown
	• For s type all characters in string are print

. or .0	• For d, i, o, u, x, X types the default precision value is used unless the value is zero in which case no characters are printed • For f, e, E types no decimal point character or digits are printed • For g or G types the precision is assumed to be 1
.n	• For d, i, o, u, x, X types then at least n digits are printed (padding with zeros if necessary) • For f, e, E types specifies the number of digits after the decimal point • For g or G types specifies the maximum number of significant digits to print For s type specifies the maximum number of characters to print

Example

```
# include <stdio.h>
/*  Objective: example of conversion specifiers */
int main()
{
   int i = 1234;
   float m = 123.525;

   printf("%7.3f  %7.2f  %7.1f   %7.0f   %7.5f\n", m, m, m, m, m);
   printf("%4d  %5.2d  %5.6d  %8.5d\n", i, i, i, i);
   printf("%3s  %12.5s  %.13s\n", "programming", "programming", "programming");

   getch();
   return 0; // indicates successful termination
} // end main
```

```
123.525  123.53    123.5      124 123.52500
1234  1234 001234     01234
programming          progr programming
Press any key to continue . . .
```

Flags

Several flags may be combined in one conversion specifier

<table>
<tr><td>-</td><td>Value is left justified (default is right justified). Overrides the 0 flag</td></tr>
<tr><td>+</td><td>Forces the sign (+ or -) to always be shown. Default is to just show the - sign. Overrides the space flag</td></tr>
<tr><td>space</td><td>Causes a positive value to display a space for the sign. Negative values still show the - sign</td></tr>
<tr><td>#</td><td>Alternate form:
<table>
<tr><th>Conversion Character</th><th>Result</th></tr>
<tr><td>o</td><td>Precision is increased to make the first digit a zero</td></tr>
<tr><td>X or x</td><td>Nonzero value will have 0x or 0X prefixed to it</td></tr>
<tr><td>E, e, f, g, or G</td><td>Result will always have a decimal point</td></tr>
<tr><td>G or g</td><td>Trailing zeros will not be removed</td></tr>
</table>
</td></tr>
<tr><td>0</td><td>For d, i, o, u, x, X, e, E, f, g, and G leading zeros are used to pad the field width instead of spaces. This is useful only with a width specifier. Precision overrides this flag</td></tr>
</table>

Example

```
#include <stdio.h>
/*  Objective: example of conversion specifiers */
int main()
{
   int i = 1234;
   float m = 123.0;

   printf("%#7.0f  %g  %#g\n", m, m, m);
   printf("%d %07d %-7d %+d %d %d\n", i, i, i, i, i, -i);
   printf("%o %#o %x %#x\n", 025, 025, 0x25, 0x25);
   printf("%+0d %09d %+09d\n", 445, 445, 445);

   getch();
   return 0; // indicates successful termination
} // end main
```

```
   123.  123  123.000
1234 0001234 1234    +1234 1234 -1234
25 025 25 0x25
+445 000000445 +00000445
Press any key ...
```

Escape sequence

ASCII Code	Escape sequence	Represents
7	\a	alert (bell)
92	\\	backslash
8	\b	backspace
63	\?	question mark
12	\f	formfeed
39	\'	single quote
10	\n	newline
34	\"	double quote
13	\r	carriage return
-	\ooo	octal number
9	\t	horizontal tab
-	\xhh	hexadecimal number
11	\v	vertical tab
0	\0	null character

Escape sequence

```
printf("1\t10\n"); // \n = code(10) code(13)
printf("2\t20\n");
printf("5\\9\n");
printf("programming\rc\n");
```

```
1       10
2       20
5\9
crogramming
Press any key to continue . . .
```

printf return value

- The return value is the number of values that were read, or the value of EOF (-1) if an input failure occurs

```
printf("\nn = %d\n", printf("C/C++"));
printf("\nn = %d\n", printf("%d", -1234));
```

```
C/C++
n = 5
-1234
n = 5
Press any key to continue . . .
```

Formatted input - scanf

- Input formatting **scanf("format", arguments);**
  - Input all types of data
  - Input specific characters
  - Skip specific characters

- **Format**: an input field is specified with a conversion specifer which begins with the % character. After the % character come the following in this order:
  - [*] Assignment suppressor (optional).
  - [width] Defines the maximum number of characters to read (optional).
  - [modifier] Overrides the size (type) of the argument (optional).
    - Similar to printf modifier
  - [type] The type of conversion to be applied (required).

  - Example: %*[**width**][**modifier**]**type**

%	*	12	L	g

- **Arguments**: pointers to variables where input will be stored (address of variables)

scanf

- These functions take input in a manner that is specified by the format argument and store each input field into the following arguments in a left to right fashion
- Each input field is specified in the format string with a conversion specifier which specifies how the input is to be stored in the appropriate variable
- Other characters in the format string specify characters that must be matched from the input, but are not stored in any of the following arguments
- If the input does not match then the function stops scanning and returns
- A whitespace character may match with any whitespace character (space, tab, carriage return, new line, vertical tab, or formfeed) or the next incompatible character

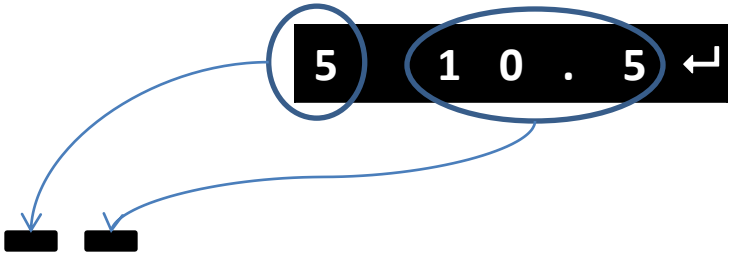
type

- It also controls what a valid convertible character
  - what kind of characters it can read so it can convert to something compatible

type	Input
d	Type signed int represented in base 10. Digits 0 through 9 and the sign (+ or -)
i	Type signed int. The base (radix) is dependent on the first two characters. If the first character is a digit from 1 to 9, then it is base 10. If the first digit is a zero and the second digit is a digit from 1 to 7, then it is base 8 (octal). If the first digit is a zero and the second character is an x or X, then it is base 16 (hexadecimal)
o	Type unsigned int. The input must be in base 8 (octal). Digits 0 through 7 only
u	Type unsigned int. The input must be in base 10 (decimal). Digits 0 through 9 only
x, X	Type unsigned int. The input must be in base 16 (hexadecimal). Digits 0 through 9 or A through Z or a through z. The characters 0x or 0X may be optionally prefixed to the value
e, E, f, g, G	Type float. Begins with an optional sign. Then one or more digits, followed by an optional decimal-point and decimal value. Finally ended with an optional signed exponent value designated with an e or E
s	Type character array. Inputs a sequence of non-whitespace characters (space, tab, carriage return, new line, vertical tab, or formfeed). The array must be large enough to hold the sequence plus a null character appended to the end
[...]	Type character array. Allows a search set of characters. Allows input of only those character encapsulated in the brackets (the scanset). If the first character is a carrot (^), then the scanset is inverted and allows any ASCII character except those specified between the brackets. On some systems a range can be specified with the dash character (-). By specifying the beginning character, a dash, and an ending character a range of characters can be included in the scan set. A null character is appended to the end of the array
c	Type character array. Inputs the number of characters specified in the width field. If no width field is specified, then 1 is assumed. No null character is appended to the array
p	Pointer to a pointer. Inputs a memory address in the same fashion of the %p type produced by the printf function.
n	The argument must be a pointer to an int. Stores the number of characters read thus far in the int. No characters are read from the input stream
%	Requires a matching % sign from the input

Example

```
int main()
{
    int i;
    float f;

    scanf("%d  %f", &i, &f);
    // space between conversion specifer is important for char type
    scanf("%d%f", &i, &f);
    printf("i = %d, f = %f", i, f);

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



Number Space Number

Number Tab Number

Number Enter Number

Example

```
int main()
{
    char c1, c2, c3;
    int i, j;
    scanf("%c  %c  %c", &c1, &c2, &c3);
    scanf("%c%c%c", &c1, &c2, &c3);

    scanf("%c -%c-%c", &c1, &c2, &c3);
    scanf("%c  - %c - %c", &c1, &c2, &c3);
    scanf("%d  t %d", &i, &j);
    scanf("%dt%d", &i, &j);
    system("PAUSE");
    return 0;
}
```

?

Example

```
int main()
{
    int i;

    scanf("%i", &i);
    printf("i = %d", i);

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

```
1 0
i = 1 0
```

```
0 1 0
i = 8
```

```
0 x 1 0
i = 1 6
```

```
0 1 3 8
i = 1 1
```

```
0 4 5 i 1
i = 3 7
```

```
// scanf("%d", &i);
```

```
0 x 1 0
i = 0
```

Example

```
int main()
{
    char c;

    scanf("%c", &c);
    printf("c = %c", c);

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

```
6  5
c  =  A
```

```
A
c  =  A
```

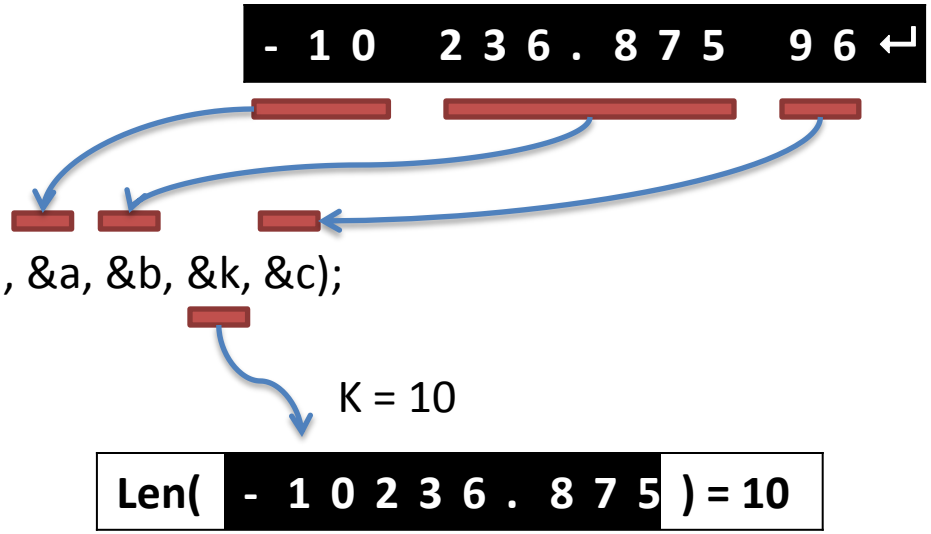
Example

```
int main()
{
    unsigned short j;
    short k;

    scanf("%u %u", &k, &j);
    printf("%u, %u", k, j);
    printf("%d, %d", k, j);

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

```
- 2     - 2
6 5 5 3 4     6 5 5 3 4
- 2     - 2
```

Example

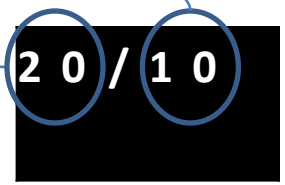
```
int main()
{
    short a, b, c, k;

    scanf("%d %d %n %d", &a, &b, &k, &c);

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



Assignment suppressor

- *: Causes the input field to be scanned but not stored in a variable

```
int x, y;
scanf("%d %*c %d", &x, &y);
```



Width

- The maximum width of the field is specified here with a decimal value
  - If the input is smaller than the width specifier (i.e. it reaches a nonconvertible character), then what was read thus far is converted and stored in the variable

Example

```
int main()
{
    int i, j, k;

    scanf("%3d %3d %3d", &i, &j, &k);

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

```
1  2  3
```

```
1 4 2  3 5 6  4 9 8
```

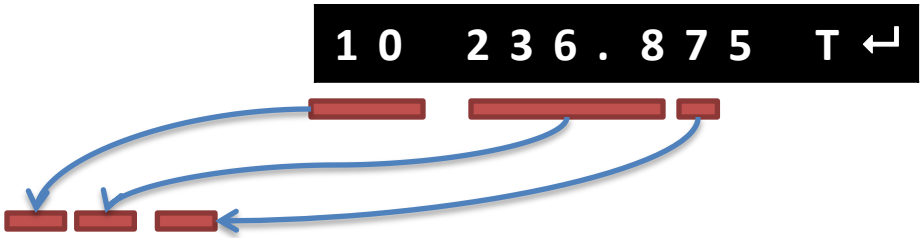
```
1 4 2 6  5 6 6 4  8
```

```
1 4 2 6 9 5 6 6 4 9 8
```

a	b	c
1	2	3
142	356	498
142	6	566
142	695	664

Example

```
int main()
{
    int i;
    float f;
    char c;

    scanf("%3d %5f %c", &i, &f, &c);

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



Example

- Consider the following code;
  - scanf ("%d", &n);
  - scanf ("%c", &op);
- Many students have done this, only to find the program "just skips through" the second scanf. Assume you type 45\n in response to the first scanf. The 45 is copied into variable n. When your program encounters the next scanf, the remaining \n is quickly copied into the variable op

scanf return value

- The scanf function returns an integer, which indicates the number of successful conversions performed
  - lets the program check whether the input stream was in the proper format

```
printf("%d", scanf("%d/%d/%d", &Month, &Day, &Year));
```

Input Stream	Return Value
02/16/69	3
02 16 69	1

Arguments

- What's wrong with the following?

```
int n = 0;
scanf("%d", n);
```

- Of course, the argument is not a pointer!
  - scanf will use the value of the argument as an address
  - What about a missing data argument?

```
scanf("%d");
```

  - scanf will get an address from stack, where it expects to find first data argument - if you're lucky, the program will crash trying to modify a restricted memory location

Arguments

- The number of arguments in a call to printf or scanf depends on the number of data items being read or written

- printf or scanf parameters pushed on the stack from right to left

  - int i = 5;
  - printf("%d %d", ++i, ++i); // 7 6
  - printf("%d %d", i++, i++); // 8 7

getchar and putchar

- get character from keyboard buffer
  - Input from the keyboard is buffered in an input buffer managed by the operating system
- **getchar()** is used to read in a character
  - No input parameter
  - Read first character after press Enter key
  - Return the int data type. The value return correspond to the ASCII code of the character read
- **putchar()** is used to display a character
  - Required the char variable as input parameter
  - Return the int data type. The value return correspond to the ASCII code of the character displayed

getch and getche

- getch()
  - Ignore keyboard buffer
  - Without echo input character

- getche()
  - Ignore keyboard buffer
  - echo input character

Example

```
printf("c = %c\n", getchar());
printf("c = %c\n", getch());
printf("c = %c\n", getche());

printf("toupper = %c\n", toupper(getche()));// ctype.h
printf("toupper = %c\n", tolower(getche()));// ctype.h
```