



## C Language

---



## Types dimensions

---

<b><u>type</u></b>	<b><u>bytes</u></b>
char	1 or 2
short	2
int	2 or 4
long	4
float	4
double	8
long double	10

- Dimension of int type depends on platform and compiler
- char type must be smaller than int type



## Example of integer constants

---

12312	int
12312u	unsigned int
0xABCU	unsigned int
12345678L	long
12345678UL	unsigned long
0x123ABCL	long
0347	int
0347L	long



## Floating-point Constants

---

- Floating-point number must contain :
  - dot or
  - exponent (es. 12e-3) or
  - dot and exponent
- A float constant is interpreted as "double" unless suffixes are used:
  - F, f => float constant
  - L, l => long double constant

## Code blocks and Scope

---

- Each block is enclosed by **braces** { } and starts a new **scope** for the variables
- All local variables declared within the block are not visible from outside the block

```
for (int i=0; i<10; i++){  
    ...  
}
```

5

## Control statements

---

- if -else,
- switch,
- while,
- do-while,
- for,
- break,
- continue

6

## Conditions

- When the condition contain an expression whose result is an int value
  - If the value is equals to 0 → condition false
  - If the value is different from 0 →condition true

```
int x = 7;
if (x) {...}
```

- Better use relational operators

```
if (x != 0)
```

7

## Operators (integer and floating-point)

- **Arithmetical**    +   -   \*   /   %
- **Relational**    >   <   >=   <=   ==   !=
  - Return 1 for true, 0 for false
- **Assignment**    =   +=   -=   \*=   /=   %=   &=   |=   ^=
- **Increment**    ++   --
  - Prefix ( ++x ) and postfix ( x++ )
- Unary operators (i.e. with one operand, like ++, --) have precedence on binary operators ( 2 operands)
- Arithmetical operators have precedence on relational operators, which have precedence on logical operators

8

## Logical operators

- Logical operators :

- `&&` (and)
- `||` (or)
- `!` (not)
- `^` (xor, i.e. exclusive-or)

- Bitwise operators work on integers interpreted as binary numbers

- `&` | `^` (and, or, xor)
- `<<` (left-shift)
- `>>` (right-shift)
- `~` (negation / complement)

9

## Logical operators

- Logical operators :

- `&&` (and) has precedence on `||` (or)
- `!` (not) is unary operator and has precedence on all binary operators

- A logical expression is evaluated from left to right and the evaluation stops as soon as the value is determined, e.g.:

- ```
int x= 3, y= 4;
if (x<0 && ++y>0) y += 2;
```
- What's the value of 'y' ?

10

## scanf function

- Can store inputs in variables from standard input (keyboard)
- **scanf(format, variables list);**
  - **format** is a string specifying input data format;
  - Variables list is the comma-separated list of variables' addresses)

## Format strings in scanf

|                 |                             |
|-----------------|-----------------------------|
| <b>%d</b>       | int (decimal)               |
| <b>%i</b>       | int (decimal, octal or hex) |
| <b>%o</b>       | int octal                   |
| <b>%x</b>       | int hex                     |
| <b>%c</b>       | char                        |
| <b>%f</b> %e %g | floating point              |
| <b>%s</b>       | string of chars             |

```
int    a,b,c; char  x,y,z;
float  f1,f2; .....
scanf("%c%c",&x, &y);
scanf("%d%f%d",&a, &f1, &b);
scanf("%c%d%f",&z, &c, &f2);
```



## printf function

---

Can print variables on standard output (screen)

**printf( format, variables list);**

**format** is a string specifying output data format;

**Variables list** is the comma-separated list of variables

```
int a=100, b=10;
```


```
.....
```

```
printf("a = %d\nc = %d\n", a, b*3);
```

This code produces this output

```
a = 100
```

```
c = 30
```



## sprintf() and sscanf()

---

- **sprintf** can print variables on a string
- **sscanf** can read variables from a string

**sprintf( line, format, variables list);**

**line** is the output string

**format** is a string specifying output data format;

**Variables list** is the comma-separated list of variables

**sscanf( line, format, variables list);**

**line** is the input string

**format** is a string specifying input data format;

**Variables list** is the comma-separated list of variables in which to store the parsed data



## Example with sprintf() - sscanf()

```
#include <stdio.h>
int main( ) {
int numbers[5] = {74, 18,33,30,97}; int result[5], index;
char line[80];
/* sprintf builds a string 'line' out of many elements */
sprintf(line,"%d %d %d %d %d\n",
    numbers[0],numbers[1],numbers[2],numbers[3],numbers[4]);
printf("%s",line);
/* sscanf parses many elements from a string 'line' */
sscanf(line,"%d %d %d %d %d",
    &result[4],&result[3],(result+2),(result+1),result);
for (index = 0;index < 5;index++)
    printf("The final result is %d\n",result[index]);
getchar();
}
```

15



## fgets(): read text from a File

```
#include "stdio.h"
main( ) {
FILE *fp1; char oneword[100]; char * c;
fp1 = fopen("TENLINES.TXT","r");
do {
    c = fgets(oneword,100,fp1); /* get one line from the file */
    if (c != NULL)
        printf("%s", oneword); /* display it on the monitor */
} while (c != NULL); /* repeat until NULL */
fclose(fp1); /* this releases Operating System resources */
}
```

16



## fprintf(): output to a file

---

```
#include "stdio.h"
main( ) {
    FILE *fp; char stuff[25]; int index;
    fp = fopen("TENLINES.TXT","w"); /* open for writing */
    strcpy(stuff,"This is an example line.");
    for (index = 1;index <= 10;index++)
        fprintf (fp,"%s Line number %d\n",stuff,index);
    fclose(fp); /* close the file before ending program */
}
```

17



## Punctuation and typography

---

Parentheses ( ) square brackets [ ] Braces { }  
 colon : comma , dot/period . semicolon ;  
 question mark ? exclamation mark !  
 quotation marks " " Apostrophe ' tilde ~  
 slash / backslash \ underscore \_ dash -  
 at sign @ asterisk/star \* Hash #  
 Ampersand & pipe | caret ^  
 Less than < more than > percent/modulo %

18