C programming language

Introduction:

- C is a general-purpose, high-level language.
- C has now become a widely used language because:

✓ Easy to learn.
✓ C is Procedural Oriented Programming language
✓ It can be compiled on a variety of computer platforms.

C language is case sensitive

Steps to write C program:

❖ You need 2 tools:

- Text editor
- Compiler

❖ To write C program:

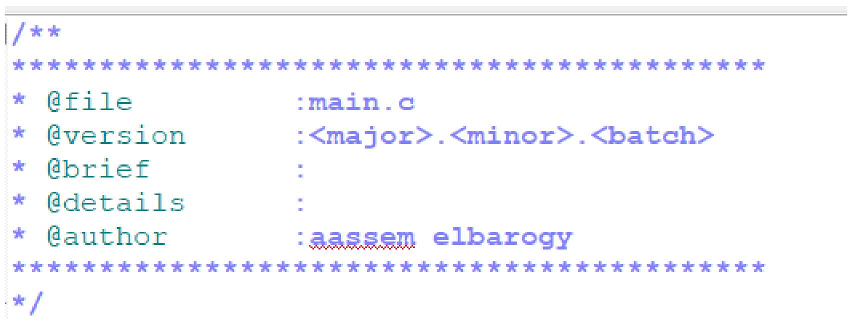
1. Create a program.
2. Compile program.
3. Run program.

Program structure:

In big companies there is rules file for coding shape.

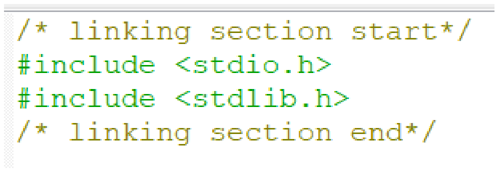
Most companies use this sequence:

1. Documentation section

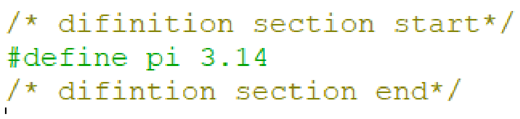


```
/**
*********************************************
* @file          :main.c
* @version       :<major>.<minor>.<batch>
* @brief         :
* @details       :
* @author        :aassem elbarogy
*********************************************
*/
```

2. Linking section
3. Definition section

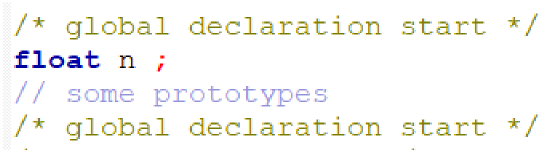


```
/* linking section start*/
#include <stdio.h>
#include <stdlib.h>
/* linking section end*/
```



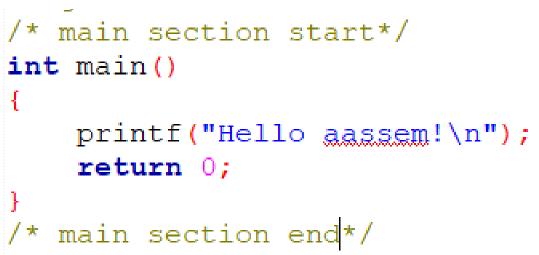
```
/* difinition section start*/
#define pi 3.14
/* difintion section end*/
```

4. Global declaration section



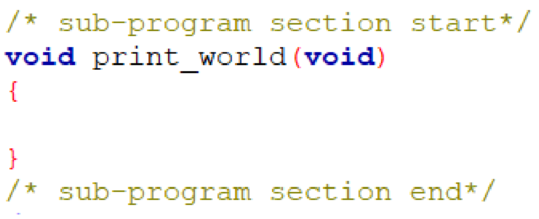
```
/* global declaration start */
float n ;
// some prototypes
/* global declaration start */
```

5. Main function section



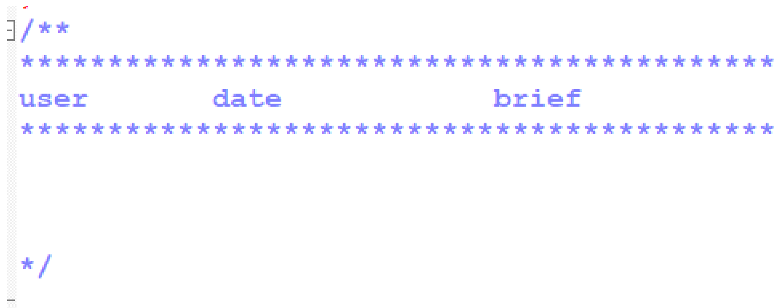
```
/* main section start*/
int main()
{
    printf("Hello aassem!\n");
    return 0;
}
/* main section end*/
```

6. Sub-program section



```
/* sub-program section start*/
void print_world(void)
{

}
/* sub-program section end*/
```

7. History log



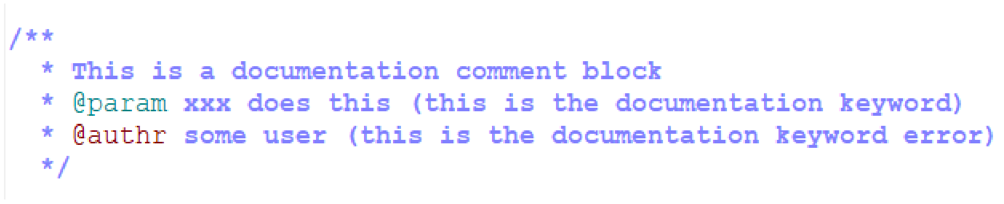
```
/**
*******************************************
user          date                brief
*******************************************



*/
```

-

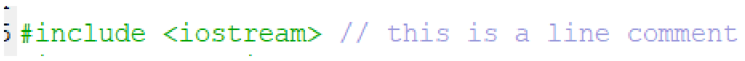
Basic syntax:

Comments:

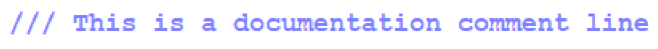
There are different types of comment and here are some examples.



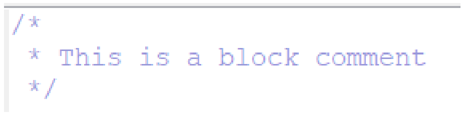
```
/**
  * This is a documentation comment block
  * @param xxx does this (this is the documentation keyword)
  * @authr some user (this is the documentation keyword error)
  */
```



```
#include <iostream> // this is a line comment
```

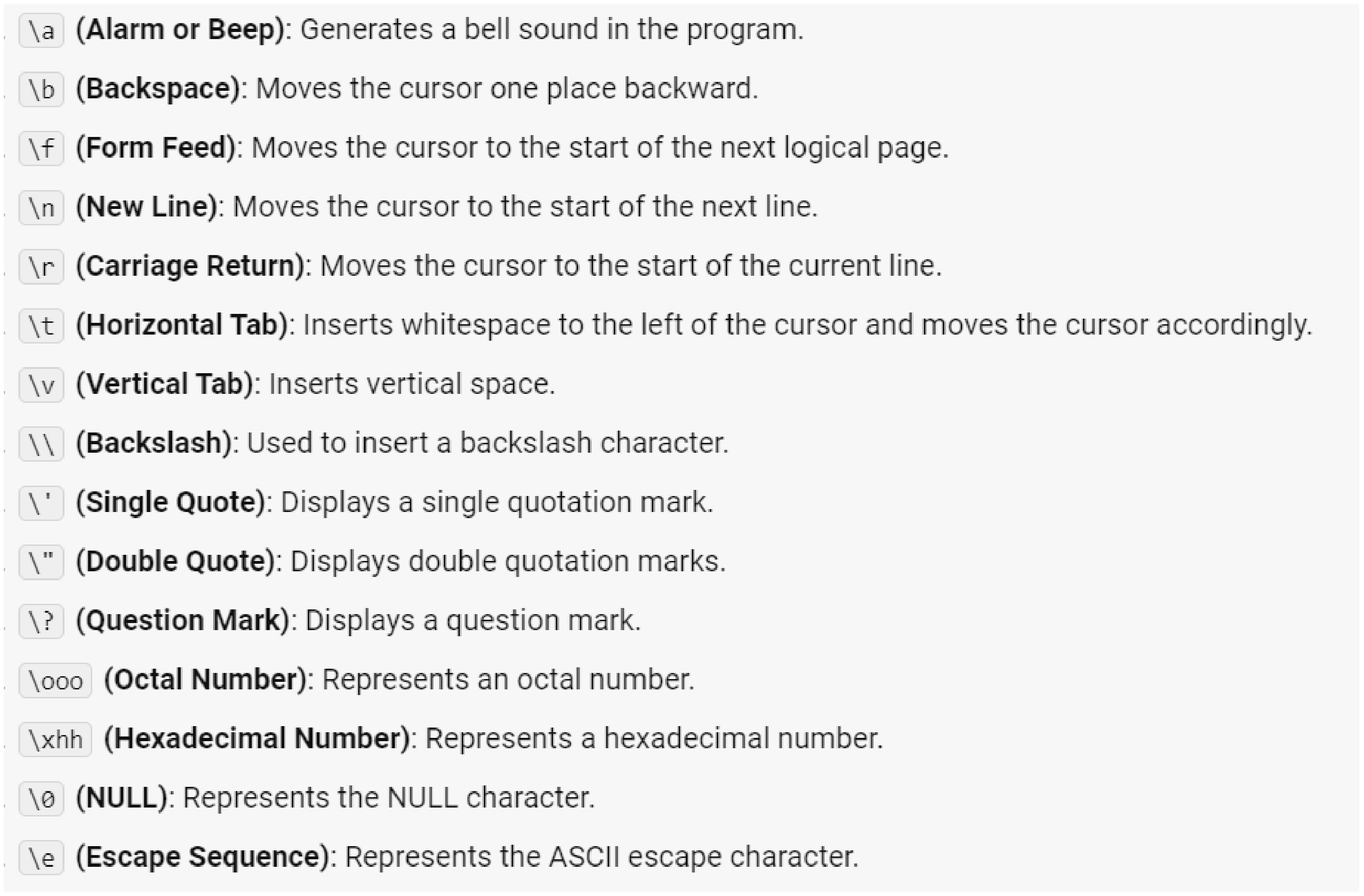


```
/// This is a documentation comment line
```



```
/*
 * This is a block comment
 */
```

Escape sequences:



- `\a` **(Alarm or Beep)**: Generates a bell sound in the program.
- `\b` **(Backspace)**: Moves the cursor one place backward.
- `\f` **(Form Feed)**: Moves the cursor to the start of the next logical page.
- `\n` **(New Line)**: Moves the cursor to the start of the next line.
- `\r` **(Carriage Return)**: Moves the cursor to the start of the current line.
- `\t` **(Horizontal Tab)**: Inserts whitespace to the left of the cursor and moves the cursor accordingly.
- `\v` **(Vertical Tab)**: Inserts vertical space.
- `\\` **(Backslash)**: Used to insert a backslash character.
- `\'` **(Single Quote)**: Displays a single quotation mark.
- `\"` **(Double Quote)**: Displays double quotation marks.
- `\?` **(Question Mark)**: Displays a question mark.
- `\ooo` **(Octal Number)**: Represents an octal number.
- `\xhh` **(Hexadecimal Number)**: Represents a hexadecimal number.
- `\0` **(NULL)**: Represents the NULL character.
- `\e` **(Escape Sequence)**: Represents the ASCII escape character.

Linking:

```
#include <liberary_name.h>
```

Functions:

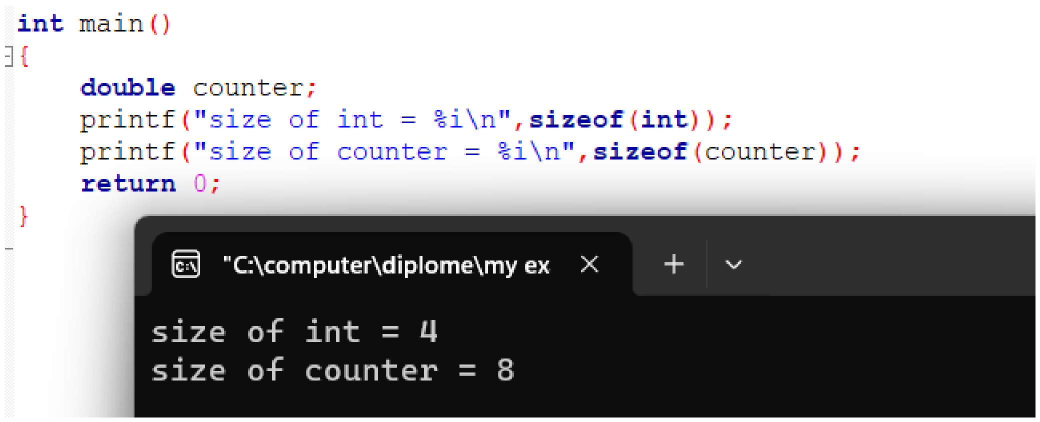
Printf:

```
printf("anything");
```

You can edit your text by using escape sequences.

Be careful when you use escape sequences to avoid errors.

Sizeof:



```
int main()
{
    double counter;
    printf("size of int = %i\n",sizeof(int));
    printf("size of counter = %i\n",sizeof(counter));
    return 0;
}
```

```
size of int = 4
size of counter = 8
```

Every statement should end with semi-colons.

Tokens:

1) Keywords
2) Constants
3) Identifiers
4) Strings
5) Operators
6) Special symbols

Keyword:

auto	break	char	const
continue	for	double	while
else	enum	void	extern

Identifier:

It is a name used to identify variables, functions or any other user-identified item.

Identifiers start with a letter or underscore.

Identifiers can’t start with numbers.

White spaces:

C compiler totally ignores blank lines.

You can’t add space between identifier characters.

There must be at least one space between keyword and identifier.

<u>Data types:</u>

❖Basic types
- Integer types
- Floating-point types

❖Enumerated types.

❖Derived types
- Pointer types
- Array types
- Structure types.
- Union types
- Function types

❖Void type: no value is available.

If there is no data type used int is default

If global variables not initialized value is zero

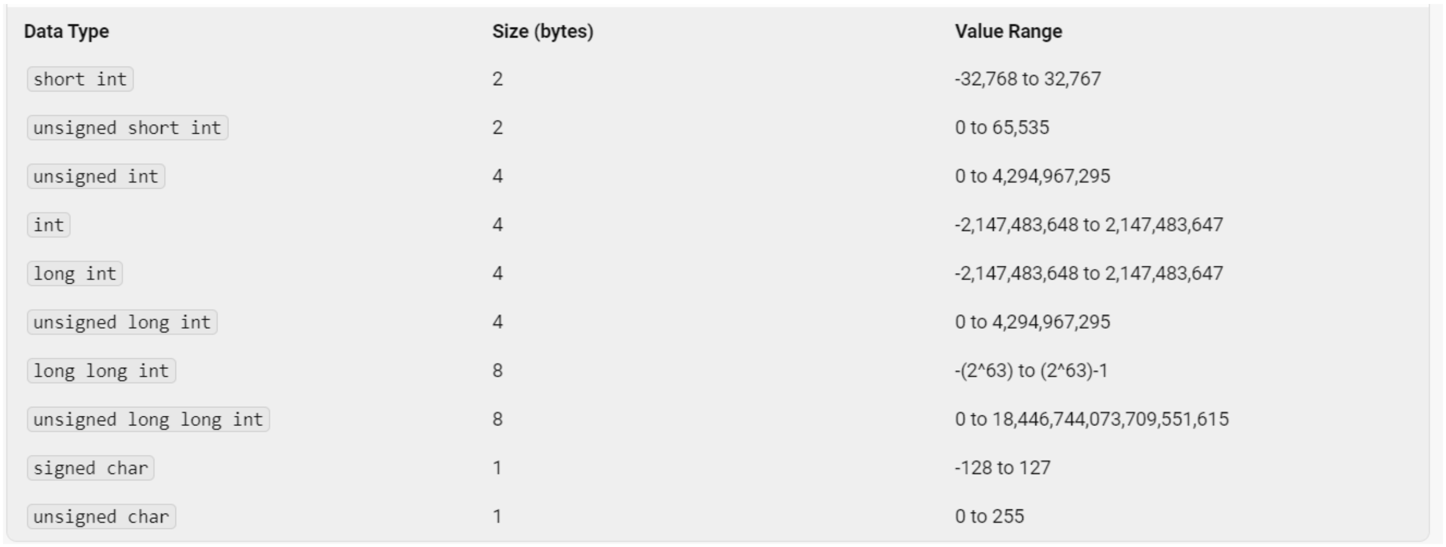
If local variables not initialized value is unknown (garbage)

You can’t change global constant variables.

You can change local constant variables by using pointers(indirect).

It doesn't matter either you put const first or data type frist

Integer data types:



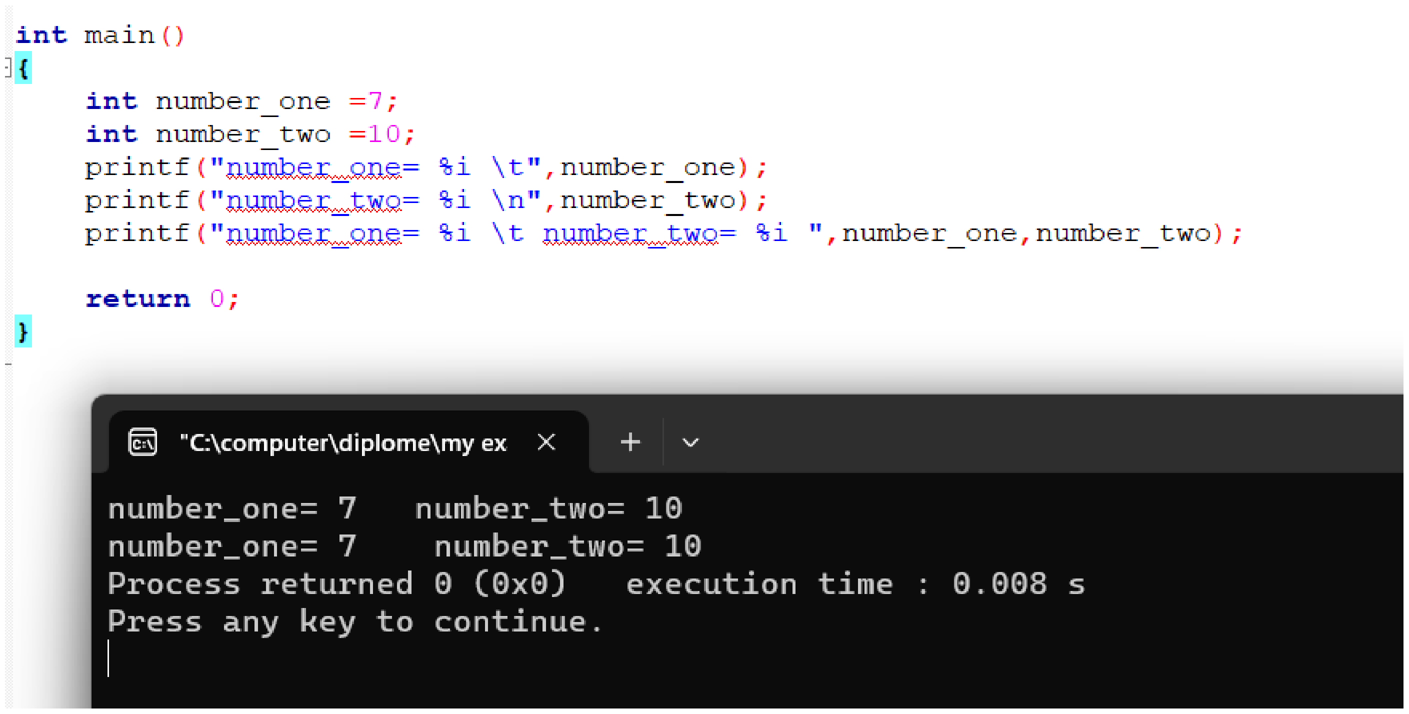
Data Type	Size (bytes)	Value Range
`short int`	2	-32,768 to 32,767
`unsigned short int`	2	0 to 65,535
`unsigned int`	4	0 to 4,294,967,295
`int`	4	-2,147,483,648 to 2,147,483,647
`long int`	4	-2,147,483,648 to 2,147,483,647
`unsigned long int`	4	0 to 4,294,967,295
`long long int`	8	-(2^63) to (2^63)-1
`unsigned long long int`	8	0 to 18,446,744,073,709,551,615
`signed char`	1	-128 to 127
`unsigned char`	1	0 to 255

You must consider the size of these types to manage memory consumption better.

Types are signed by default.

The unsigned means type is in positive region.

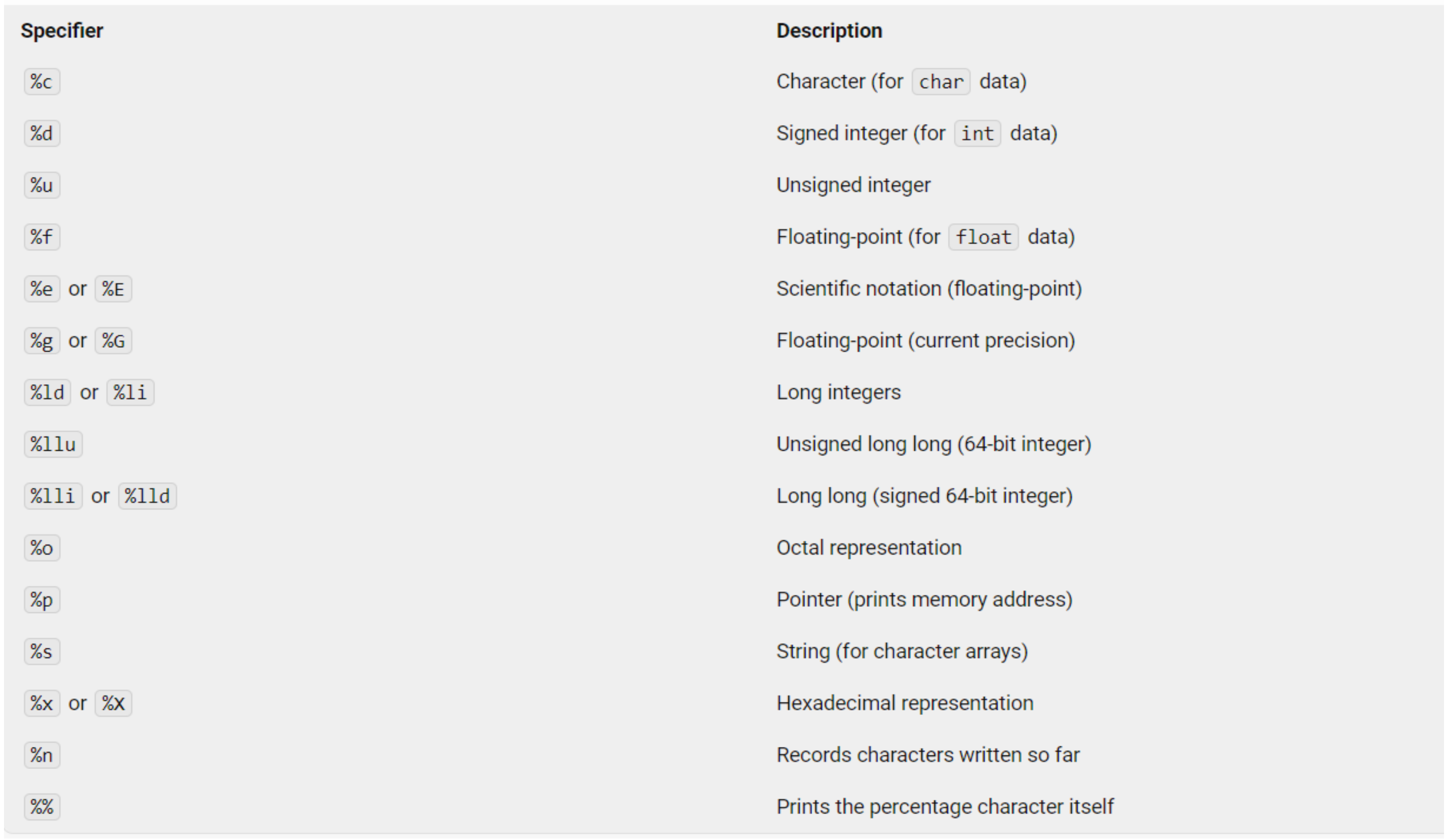
Data specifier:



```
int main()
{
    int number_one =7;
    int number_two =10;
    printf("number_one= %i \t",number_one);
    printf("number_two= %i \n",number_two);
    printf("number_one= %i \t number_two= %i ",number_one,number_two);

    return 0;
}
```

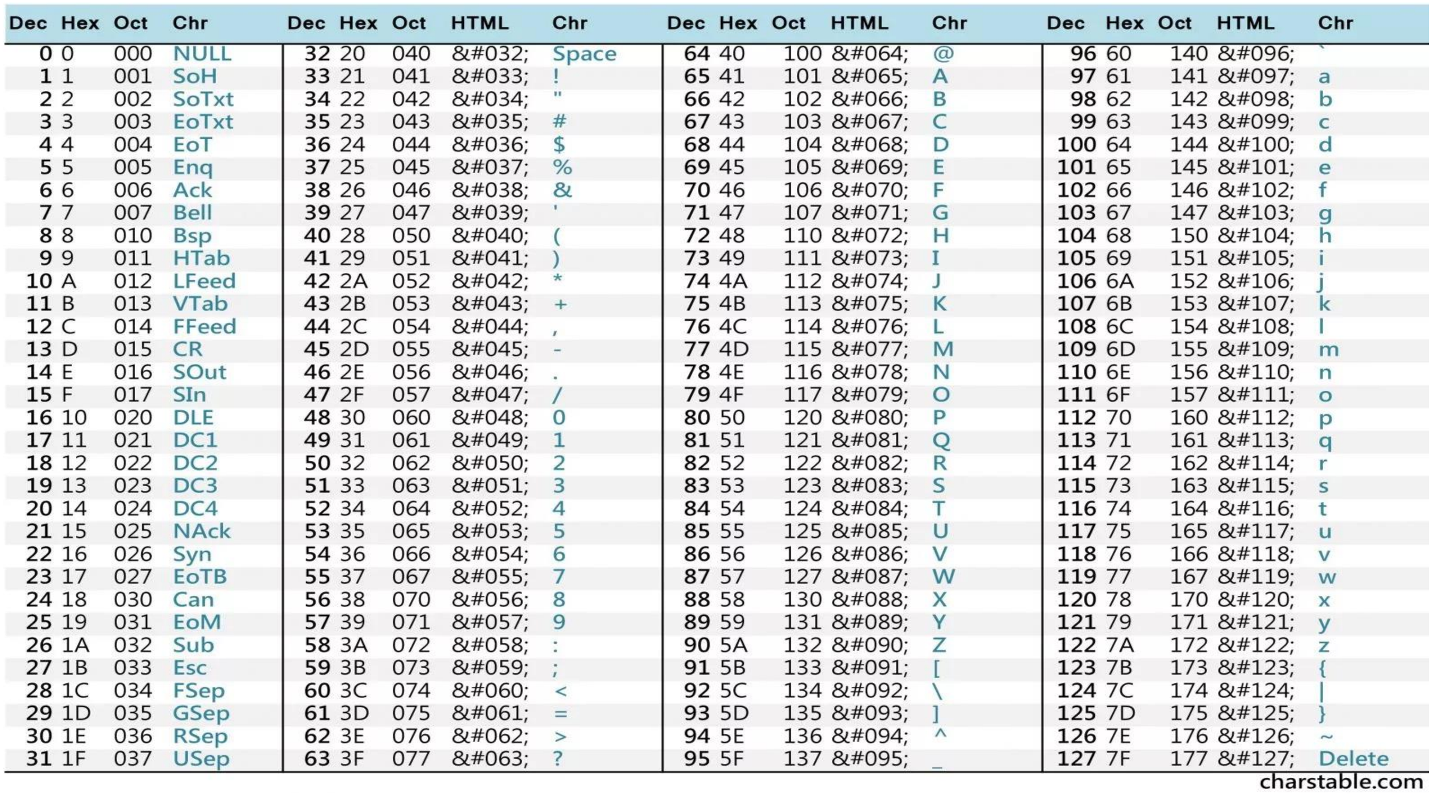
```
number_one= 7   number_two= 10
number_one= 7    number_two= 10
Process returned 0 (0x0)   execution time : 0.008 s
Press any key to continue.
```

Other specifiers:



Specifier	Description
`%c`	Character (for `char` data)
`%d`	Signed integer (for `int` data)
`%u`	Unsigned integer
`%f`	Floating-point (for `float` data)
`%e` or `%E`	Scientific notation (floating-point)
`%g` or `%G`	Floating-point (current precision)
`%ld` or `%li`	Long integers
`%llu`	Unsigned long long (64-bit integer)
`%lli` or `%lld`	Long long (signed 64-bit integer)
`%o`	Octal representation
`%p`	Pointer (prints memory address)
`%s`	String (for character arrays)
`%x` or `%X`	Hexadecimal representation
`%n`	Records characters written so far
`%%`	Prints the percentage character itself

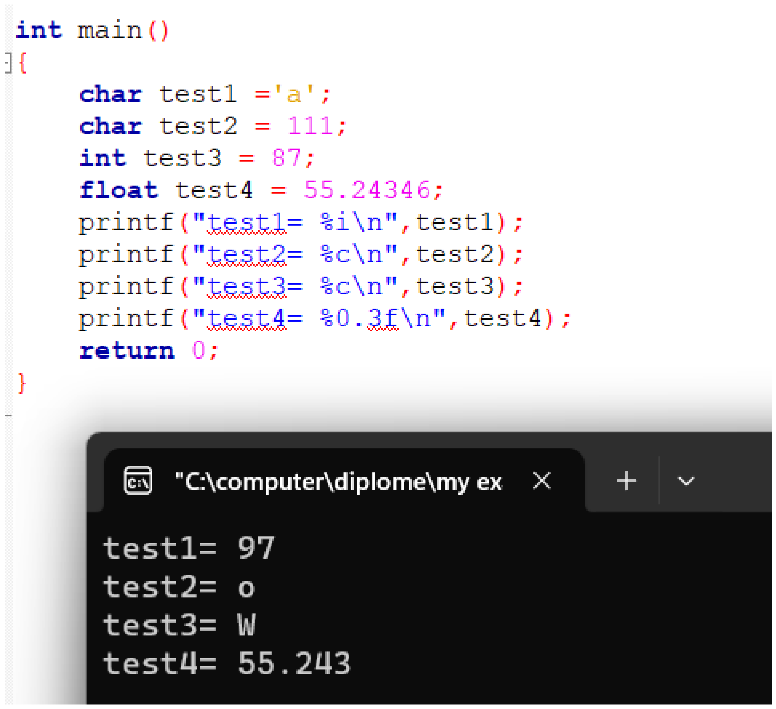
- Every character has an ascii code.



Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	SoH	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	SoTxt	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	EoTxt	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	EoT	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enq	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Ack	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Bsp	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	HTab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	LFeed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	VTab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	FFeed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	CR	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	SOut	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	SIn	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	DLE	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	DC1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	DC2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	DC3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	DC4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	NAck	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Syn	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	EoTB	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Can	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	EoM	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Sub	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Esc	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	FSep	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	GSep	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	RSep	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	USep	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Delete

charstable.com

- When you use float specifiers you can limit number of digits
- You can use character variables as characters or numbers.



```c
int main()
{
    char test1 ='a';
    char test2 = 111;
    int test3 = 87;
    float test4 = 55.24346;
    printf("test1= %i\n",test1);
    printf("test2= %c\n",test2);
    printf("test3= %c\n",test3);
    printf("test4= %0.3f\n",test4);
    return 0;
}
```

```
test1= 97
test2= o
test3= W
test4= 55.243
```

<u>Memory allocation:</u>

Declaration: Provides basic information, no memory allocation. When using “extern”.

Definition: Specifies details and allocates memory