In The Name Of God
Principles Of Programming
(Session 2)
Chapter 1
K&R

Presenter : Graders' Team – Spring 2019

Some fundamentals :

1 bit -> either 0 or 1

1 byte = 8 bits = xxxx xxxx

We can make 2^8 = 256 different numbers with 8 bits

1 character = 1 byte = 8 bits -> which means -> 1 char can be used to store a non-negative number from 0 to 255 .

ASCII Table :

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	≡
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99n	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

C is a ‘ Typed’ language

Char -> character - a single byte

int -> integer

Short -> short integer

Long -> double size of an int (might vary in different platforms)

Double -> double size of a long (might vary in different platforms)

Main Function

```
int main(void) {
        // write your code here
        Return 0 ;
 }

 Void main (void){
        //write your code here
        return 0 ; // some times the compiler doesn't get this  as an error .
        // but you should not put it at the end of any void function.
 }
 // why void as an argument ? What is that for ?
 // we call it a " command Line Argument " which we will learn later .
```

Pre – Processors :

```
#include <stdio.h>

#define   LOWER  0       /* lower limit of table */
#define   UPPER  300     /* upper limit */
#define   STEP   20      /* step size */

/* print Fahrenheit-Celsius table */
main()
{
    int fahr;

    for (fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

will be replaced by the corresponding *replacement text.*

Input & Output :

- Basic Functions in <stdio.h> :

```
int getchar (void)

void putchar (int ASCII )
// what is ‘ void ‘ ?

? printf ( character sequence , variable sequence )
// what is the return type of printf() ????
Int
// why int ?
// it returns the number of characters it has printed
```

What is the output of this code ?

```
int a  = 1000 ;
printf ( “%d” , pritnf ( “\n%d” , a ) ) ;
```

5 - why ??

{ ‘\n’ , ‘1’ , ‘0’ , ‘0’ , ‘0’ }

Escape Characters :

\a	alert (bell) character	\\	backslash
\b	backspace	\?	question mark
\f	formfeed	\'	single quote
\n	newline	\"	double quote
\r	carriage return	*ooo*	octal number
\t	horizontal tab	\x*hh*	hexadecimal number
\v	vertical tab		

Commenting in C :

inline Comments : // equivalent with # in python

Multi – line : /* bla bla bla */ equivalent with ‘’’ in python

ARLO : Arithmetic , Relational and Logic Opearators

- Arithmetic : + , - , * , /
- // C does not support any power operators (^ is in python not in C)
- // C uses Math.h library instead to support extra functions .

- Relational : < , > , <= , >= , == , !=
- # The relational operators have lower precedence than arithmetic operators which means :
- x < y -1 would be taken as x < (y – 1)
- Logical : && , || (equivalent with and , or in python)

Bitwise Operators :

C provides six operators for bit manipulation; these may only be applied to integral operands, that is, `char`, `short`, `int`, and `long`, whether signed or unsigned.

`&`	bitwise AND
`|`	bitwise inclusive OR
`^`	bitwise exclusive OR
`<<`	left shift
`>>`	right shift
`~`	one's complement (unary)

Precedence :

TABLE 2-1. PRECEDENCE AND ASSOCIATIVITY OF OPERATORS

OPERATORS	ASSOCIATIVITY
`()  []  ->  .`	left to right
`! ~ ++ -- + - * & (type) sizeof`	right to left
`*  /  %`	left to right
`+  -`	left to right
`<<  >>`	left to right
`<  <=  >  >=`	left to right
`==  !=`	left to right
`&`	left to right
`^`	left to right
`¦`	left to right
`&&`	left to right
`¦¦`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= ¦= <<= >>=`	right to left
`,`	left to right

Unary +, –, and * have higher precedence than the binary forms.

What is Piping ???

- Google it ☺

- In computer programming, especially in UNIX operating systems, a pipe is a technique for passing information from one program process to another. Unlike other forms of interprocess communication (IPC), a pipe is one-way communication only. Basically, a pipe passes a parameter such as the output of one process to another process which accepts it as input. The system temporarily holds the piped information until it is read by the receiving process.

Arrays :

=> Type name [size] ;

The Integer Array :

```
int numbers [10] ; // the array would be filled with garbage
```

OR

```
int  numbers [ ] = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 }
```

OR

```
int len = 10 ;
int numbers [len] ;
```

The Character Arrays :

```
char sentence [10] ;
OR
char sentence  [6]  = { ‘a’ , ‘b’ , ‘c’ ,  ‘d’ , ‘\n’  , ‘\0’ }
 ‘\0’ is a character that has an ASCII code of zero .
OR
char sentence [ ] = { ‘a’ , ‘b’ , ‘c’ , ‘\0’ } ;
char sentence [ ] =  “abc” ;
char sentence [ ] = { “abc” } ;
// in the last two declerations , it puts ‘\0’ automatically
//Q: What is the difference between “a” and ‘a’ ?
```

The `%s` format specification in `printf` expects the corresponding argument to be a string represented in this form. `copy` also relies on the fact that its input argument is terminated by `'\0'`, and it copies this character into the output argument. (All of this implies that `'\0'` is not a part of normal text.)

Functions :

<return type > < function name > (< function argument > , ….) {

// code

}

EX : adding 2 numbers and returning the result :

```
int adder ( int a , int b ) {
        int sum = a + b ;
        return sum ;
}
```

```
int adder ( int a , int b ) {
        return ( a + b ) ;
}
```

```
void print_result ( int a , int b ){
        int sum = a + b ;
        printf("%d" , sum );
}
```

Points To Be Mentioned :

- Declare (not necessarily define) before usage
- Function Prototype
- Local Variables
- Scope Rule
- Passing Arrays to functions (AKA call by value/reference)
- Sizeof() function
- Strlen() function
- Local Variables
- Global Variables