Section 1

Introduction to programming with C

Slide contents follow

Orhan Gazi - Modern C Programming

Prepared by:

AbdElrahman Rabea, Eng.

Why learning C

- C is programming language that is lower-level than most other languages; that means it creates code that’s a lot closer to what machines really understand.
- Improves understanding: understanding C gives you a much better idea of what’s really going on.
- Enhances other languages: Once you understand C, learning other languages (like Python, Java, or Rust) becomes easier.
- Efficiency: C produces small, fast programs with minimal overhead.
- C is used where speed, space, and portability are important.
- Most operating systems are written in C.
- Most other computer languages are also written in C.

How C works

Computers really only understand one language: machine code, a binary stream of 1s and 0s. You convert your C code into machine code with the aid of a compiler.

This happens in different steps:

- Preprocessing
- Compilation
- Assembly
- Linking

Too much headache for now

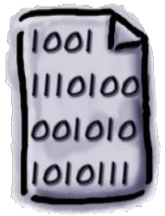
```
#include <stdio.h>
int main()
{
  puts("C Rocks!");
  return 0;
}
```

Source

You start off by creating a source file. The source file contains human-readable C code.

Compile

You run your source code through a compiler. The compiler checks for errors, and once it's happy, it compiles the source code.



Output

The compiler creates a new file called an *executable*. This file contains machine code, a stream of 1s and 0s that the computer understands. And that's the program you can run.

Lets write our first c program

1. #include <stdio.h> : a preprocessor directive that tells the compiler to include the Standard Input Output (stdio) library.
   stdio.h contains functions like printf() and scanf() for input and output operations.
2. int main(): the main function where program execution starts.
3. printf() is a function from stdio.h that prints text to the console.

```c
#include <stdio.h>
int main() {
        printf("Hello World!");
}
```

Data types

Data Type	Size	range	stores
char	1 byte (8 bits)	-128 to 127 (signed) 0 to 255 (unsigned)	Single character (e.g., 'A', 'b', '@')
double	8 bytes (64 bits)	~15-16 decimal places	Double-precision decimal numbers (e.g., 3.1415926535)
int	4 bytes (32 bits)	-2,147,483,648 to 2,147,483,647 (signed)	Whole numbers (e.g., 10, -5, 1000)
float	4 bytes (32 bits)	~6-7 decimal places	Single-precision decimal numbers (e.g., 3.14, -0.01)

variable declaration:

dataType variable_name;

int x;

```c
int main() {
    char grade = 'A';
    int age = 25;
    float price = 19.99;
    double pi = 3.1415926535;

    printf("Character: %c\n", grade);
    printf("Integer: %d\n", age);
    printf("Float: %.2f\n", price);
    printf("Double: %.10lf\n", pi);
    return 0;
}
```

More on chars

1. char is stored as an integer (ASCII value).
    1. 'A' is stored as 65 in memory.
    2. 'B' is 66, 'C' is 67, etc.
2. Printing ASCII values and character operations:
    1. %c prints the character.
    2. %d prints the ASCII integer value.
    3. grade + 1 moves to the next character in ASCII (e.g., 'A' → 'B').

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

```c
int main() {
    char grade = 'A';

    printf("Character: %c\n", grade);
    printf("ASCII Value: %d\n", grade);
    printf("Next Character: %c\n", grade + 1);
    printf("Next ASCII Value: %d\n", grade + 1);

    return 0;
}
```

Basic operators

Operator	Type	Description	Example (a = 10, b = 3)	Result
+	Arithmetic	Addition	a + b	10 + 3 = 13
-	Arithmetic	Subtraction	a - b	10 - 3 = 7
*	Arithmetic	Multiplication	a * b	10 * 3 = 30
/	Arithmetic	Division (Quotient)	a / b	10 / 3 = 3 *(integer division)*
%	Arithmetic	Modulus (Remainder)	a % b	10 % 3 = 1
&	Bitwise	AND (sets bits only if both are 1)	a & b	10 & 3 = 2 *(1010 & 0011 = 0010)*
|	Bitwise	OR (sets bits if either is 1)	a | b	10 | 3 = 11 (1010 | 0011 = 1011)
^	Bitwise	XOR (sets bits if different)	a ^ b	10 ^ 3 = 9 *(1010 ^ 0011 = 1001)*
~	Bitwise	NOT (inverts bits)	~a	~10 = -11
<<	Bitwise	Left Shift (multiplies by 2^n)	a << 1	10 << 1 = 20 *(1010 → 10100)*
>>	Bitwise	Right Shift (divides by 2^n)	a >> 1	10 >> 1 = 5 *(1010 → 0101)*

Problem: Declare two integers, perform +, -, *, /, and %, then print the results.

```c
#include <stdio.h>
int main() {
    int a = 10, b = 3;
    printf("a + b = %d\n", a + b);
    printf("a - b = %d\n", a - b);
    printf("a * b = %d\n", a * b);
    printf("a / b = %d\n", a / b);  // Integer division
    printf("a / b = %.2f\n", (float)a / b);
    printf("a % b = %d\n", a % b);
    return 0;
}
```

Swapping Two Numbers

- Challenge: can you do it with bitwise XOR operations ^?
- Show it to me next section :)

- There is a hint in the next slide (try solving it without the hint)

```c
#include <stdio.h>
int main() {
    int a = 10, b = 3;
    printf("a:%d b:%d\n", a, b); // a:10 b:3

    // swapping with temp variable
    int temp = a;
    a = b;
    b = temp;
    printf("a:%d b:%d\n", a, b); // a:3 b:10

    // swapping without temp variable
    a = a + b; // 3 + 10 = 13
    b = a - b; // 13 - 10 = 3;
    a = a - b; // 13 - 3 = 10
    printf("a:%d b:%d\n", a, b); // a:10 b:3
    return 0;
}
```

Swapping Two Numbers

- Challenge: can you do it with XOR bitwise operations?
- Show it to me next section :)

- For 2 number a , b try (a ^ b ^ a) what is the result 🤔 🤔

```c
#include <stdio.h>
int main() {
    int a = 10, b = 3;
    printf("a:%d b:%d\n", a, b); // a:10 b:3

    // swapping with temp variable
    int temp = a;
    a = b;
    b = temp;
    printf("a:%d b:%d\n", a, b); // a:3 b:10

    // swapping without temp variable
    a = a + b; // 3 + 10 = 13
    b = a - b; // 13 - 10 = 3;
    a = a - b; // 13 - 3 = 10
    printf("a:%d b:%d\n", a, b); // a:10 b:3
    return 0;
}
```

Convert an uppercase letter to lowercase without using tolower().

- Challenge2: Can you solve It using bitwise OR (|)?

- Challenge3: Can you convert an uppercase letter to lowercase or vice versa in just one line?.
  - Input ‘a’ => output: ‘A’
  - Input ‘A’ => output: ‘a’
  - Note: it is only one line that makes both conversions (a => A and A => a)

```c
#include <stdio.h>
int main() {
    char A = 'A';
    printf("letter: %c", A + 32 );
    return 0;
}
```