

[Cheat Sheet] - C

Designing code/Solving problems

1. Know and understand your problem completely.
2. Decide what data you need to solve your problem.
3. Develop an algorithm.
4. Code your solution, little by little.
5. Test your solution, little by little.

Understanding code problems:

- loop problems -> create a table with all variables
- Start at the beginning, make notes
- use e-pencils to draw in the problem to create a connection string

Preprocessor Directives

- #include -> import libraries
- #define
 - Constant = NAME value
 - Macro = NAME(var1, var2) action

Data Types:

- int
 - 1
 - 4 bits of memory
- float
 - 1.0
 - 4 bits of memory

- double
 - double precision float
 - divisible by 0
 - 8 bits of memory
- char
 - 'A';
 - single character
 - 1 bit of memory
- string
 - Array of characters
 - Buffer overflow -> when memory is damaged due an string without a null in the end (is passed by the compiler)
 - String[number of characters+1] +1 is for the null
 - H E L L O \0 = "HELLO"
 - List[number of strings][size+1]
 - To write stuff from right to left, start with an "-" -> (%-s)
 - Check string.h for more functions involving strings

Conditions

- if -

```
if(5<10){
...
}
```

- || - or (if one of the conditions is met, the statement is true)
- && - and (both conditions must be true for the statement to be true)

- else -

```
else{
...
}
```

- Exception to the if statement
- else if -

```
else if(condition){  
    ...  
}
```

- If the first if statement is false, else if is the next condition to be checked
- switch - multiple if else's

```
switch(operation){  
case 1:  
    break; --> with break, the switch statement stops  
case 2:  
    printf(...) --> without a break statement, the program continues to case  
3, until reaching a break  
case 3:  
default:  
}
```

Inputs

getchar() -

scanf("") -

Loops

- while - normal loop
 - loop runs until the condition is false

```
while(condition){  
    ...  
}
```

- do
 - run loop at least 1 time

```
do{
```

```
...
}while(condition);
```

- for
 - when you know how many times the program will run

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

- sentinel
 - when user enters an specific input (constant)

```
while(scanf != SENTINEL){
...
scanf(""); --> if the user inputs the sentinel, the program ends
}
```

Functions

- Void - returns nothing
- int/char/double/struct - returns that datatype

```
void function_name(datatype parameters){
...
}

int function_name(datatype parameter){
...
return 1;
}
```

"Hacks"

- flags
 - breaks the loop when n reaches 0

- Uses the concept of accumulators -> after the completion of a loop or condition, the accumulator changes

```
int acc = 0;
int wind = 1;
int rain = 0;
int hot = 0;

if(wind == 1)
    acc++;
if(rain == 1)
    acc++;
if(hot == 1)
    acc++;
if(acc>3)
    printf("Cancel operation");
```

- nested for loops ("1 pra vc, 1 pra mim, 2 pra vc, 1 2 pra mim, ...")

```
for(int i = 0; i<x; i++){
    for(int j = 0; j<y; j++){
        ...
    }
}
```

- Selection statements are for branching
- `int x%10` -> highlights the last digit
- `int x/10` -> deletes the last digit

Pointers

- where data is stored
- `&` is for address
- `*` is for value on the address
- get multiple data at the same time

Arrays

- String is an array
- "Pointers are letters, vectors(arrays) are line, matrix is a page, a cube is a book and a tesseract is a pile of books"
- `array[size] = {element1, element2};`
- `array[y] = 1;`
- to print an array, use a nested loop(i for rows and j for cols)
- **list of strings**

```
char str[number of items(starting at 1)][max size of the largest string];
```

Structures

- "A cadeira ideal"

```
typedef struct{  
    ...  
}[Struct_Name];
```

- Basic functions related to structures
 - Create a object

```
[Struct] create_[struct](datatype a, datatype b, char c[]){  
    'Struct' s;  
    s.a = a;  
    s.b = b;  
    strcpy(s.c, c);  
    return s;  
}  
  
[Struct] empty_[struct]()(  
    [Struct' s;  
    s.a = 0;  
    s.b = 0;  
    strcpy(s.c, "");
```

```
    return s;
}
```

- Print object

```
void print_[struct]([Struct][s]){
    printf("", p.a, p.b, p.c, ...);
}
```

- Create a array of struct

```
[Struct_Array] create_[struct_array]() {
    Struct_array = sa;
    sa.count = 0;
    sa.capacity = CAPACITY --> constant CAPACITY
    return sa;
}
```

- Print an struct array

```
void print_accesspoint_array(AccessPoint_Array apa){
    for(int i = 0; i < apa.count; i++)
        print_accesspoint(apa.ap[i]);
}
```

- Inserting a Struct object into a struct array

```
void insert_accesspoint(AccessPoint_Array *apa, AccessPoint ap){
    apa->ap[apa->count] = ap;
    apa->count += 1;
}
```

-

Files

- Files are case sensitive (FILE !=File)

```
FILE *file = fopen(filename, "r"); --> 'r' is for read, 'w' is for writting
```

- Read the whole file (open file in read mode)

```

if (file != NULL){ --> checks if the file exists
    while(fgets(buffer, buffer_size, file) != NULL){ --> reads the end of
file
        buffer[strlen(buffer)-1] = '\0';
        printf("|%s|", buffer);
    }
fclose(file); --> closes the file
}

```

- write in file(open file in write mode)

```

if (file != NULL){ --> checks if the file exists
    fprintf("...");
}
fclose(file); --> closes the file
}

```

- tokenizing strings

```

char buffer[2200];
char *ptr;
int age;
double income;

if (file != NULL){ --> checks if the file exists
    while(fgets(buffer, buffer_size, file) != NULL){ --> reads the end of
file
        buffer[strlen(buffer)-1] = '\0';

        ptr = strtok(buffer, ",") --> stop at what character? (in this case: ",")
        printf("|%s|", ptr);
        age = atoi(ptr); --> transforms a string into an integer

        ptr = strtok(NULL, ",") --> now, the pointer starts at NULL due the last
string tokenization
        income = atof(ptr) -> transforms a string into a double
        printf("|%s|", ptr);
    }
}

```



```
}  
fclose(file); --> closes the file  
}
```

Common mistakes

- assign arrays (a[] = b[];)

Miscellaneous

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(88	58	1011000	130	X					
41	29	101001	51)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	.	93	5D	1011101	135]					
46	2E	101110	56	,	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

()
 ++ -- (+) (-) !
 * / %
 + -
 > >= < <=
 == !=
 &&
 ||
 = += -= *= /= %=

ORDER



Pointers - variables that store address of another variable

P → address
**P → value at address*

```

int a; ←
int *P; ←
P = &a;
a = 5;
Print P // 204
Print &a // 204
Print &P // 64
print *P // 5 ⇒ dereferencing
*P = 8
Print a // 8
  
```

Memory

Memory layout:
 Address 204: a = 8
 Address 64: P = 204

□ → POINTER

▭ → VECTOR (ARRAY)

▩ → MATRIX

Prefix	Scientific Notation		Decimal
Kilo	10 ³		1,000
Mega	10 ⁶		1,000,000
Giga	10 ⁹		1,000,000,000
Tera	10 ¹²		1,000,000,000,000
Peta	10 ¹⁵		1,000,000,000,000,000
Exa	10 ¹⁸		1,000,000,000,000,000,000
Zetta	10 ²¹		1,000,000,000,000,000,000,000
Yotta	10 ²⁴		1,000,000,000,000,000,000,000,000

Sample Code

```
#include <stdio.h>

int main(){
    printf("Hello World!\n");

    return 0;
}
```

C Book:



C Textbook.pdf

43 MB