Basic C program Structure

include. "Controller.h" → i/o. Port/reg names /addresses to use.

uint Count, bob; global. (static) variables. placed in RAM.

/* f^n definitions */
int function1. (char x) //Parameter. x passed to the f; fun returns an. integer value
{
int i, j; // local. (automatic) variables – allocated to stack or register's.
}
– instruction's to implement the f^n.

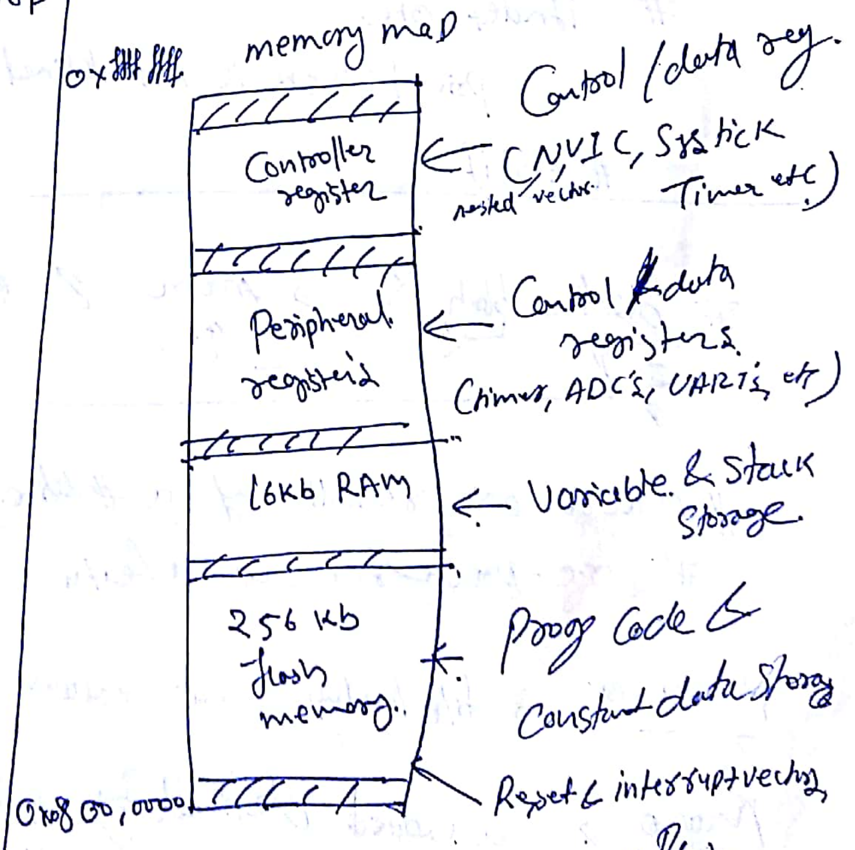
/* Main prog */

~~Ded~~ Declare. local variable.
Void main (void.) {
unsigned char sw1; } local (automatic) variable (stack or reg)
uint k;

initialize var/ device.
/* Initialization Section*/
instruction's to initialize. Variables, I/O ports, devices, f^n reg.

/* Endless loop

Body of the prog.
while (1) {
}
}



0x ffff ffff
memory map
Control /data reg.
Controller register
(NVIC, SysTick Timer etc)
nested vector
Peripheral register's
Control & data registers.
(timers, ADC's, UART's, etc)
16kb RAM
Variable. & Stack Storage.
256 kb flash memory.
Progr Code & Constant data Storage
Reset & interrupt vectors
0x08 00, 0000
1st words of flash.

NVIC → 240 - reprioritized.

The pre processor :

⇒ Resolves #defines statement's (constants, vartype, macro's.)

⇒ Concatenates #include files & Source file into 1 large file.

⇒ Processes #ifdef → #endif statements

⇒ Processes #if – # ~~ed~~ endif Statements.

⇒ for E.S. pre-processor also processes Vendor-specific directives (non-ANSI) # progma.

```
# define one 0
# ifdef one.
        printf ("one is defined");
# endif.
# ifndef one.
        printf ("one is not defined");
# endif.
```

$$\begin{tikzcd}\underbrace{\text{\# define}} & \underbrace{\text{blah}} \arrow[r] & 8 \rightarrow \text{macro} \quad \& \ \# \text{ is pre-processor directive}\end{tikzcd}$$

macro are result of a #define statement.
pre-processor is a feature of C.

Pre processor → file inclusion, macro expansion, conditional. Compilation

Macro ⇒ a word defined by #define pre-processor directive that evaluates to some one's expression's.

C - compiler data types.

Data type example.

Read bit from GPIOA. — uint16_t n; n = GPIOA -> IDR.

write TM2. value.

— Uint16_t t; TIM2 -> PSC = t.

Read 32 bit value from ADC — Uint 32_t a; a = ADC..

System Control. value. sany

— int32_t Ctrl, Ctrl = (x+y)*z;

<u>Variable</u> :- is an addressable Storage location. to information to be used by the Program.

-> each. variable must be declared to indicate Size. and type. of information to be. Stored, plus name. to be used to reference the info

Array :- set of data, Stored in consecutive. memory locations, beginning at a named address

-> Declare array name & no. of data elements, N.

int n [5)

C - operator's and their Arithmetic operation.

!	Logical negation.
^	Bitwise negation
&&	logical AND
&	Bitwise AND, address of variable or Structure.
||	logical -OR
|	Bitwise OR.
^	Bitwise XOR
+	Add^n.
++	increment.
—	Sub^n
--	Decrement
*	Mul^n, indirection (Pointer to variable)
/	Division.
%	modulus
==	Conditional equals.
!=	-11- not equals.
<	Conditional less than

<= —— Condt^{nal} less than or equals to

~~<<~~ << —— Shift left.

> — Condt. Greater than.

>=

>> —— Shift right.

C - Compound Assignment Statement Operator's.

Arthmt
Assignmt oprs.

&= —— And with assignment variable & store the result in the assignment variable.

~~a*~~ a+=b
a = a+b

|= —— OR with the assignment — " ——.

^= —— XOR with the assignmt var — " ——.

+= $\longrightarrow$ Add to the assignmt Variable.

-= —— SUBTRACT from the assignmt Var.

*= —— MUL —— " ——

/= —— Div —— " ——.

%= —— MODULUS —— " ——.

<<= —— Shift —— " —— left.

>>= —— Shift —— " —— right.

C "Backslash" Characters.

\r	→	Carriage return (CR)
\n	→	Line feed (LF)
\f.	——	form feed (FF)
\b	——	Backspace (BS.)
\t	——	Horizontal tab (HT)
\v	——	Vertical tab (VT)
\a	——	Alarmbell (BEL)
\'	——	Single quote (')
\"	——	Double quote
\\	——	Backslash. (\)
\ddd	——	Octal Number
\xddd	——	Hexadecimal Charat.

• following words can't be used in C appⁿ as labels:-

break	goto
case	if
continue	return
default	switch
do	while.
else	
for	

C- operator	order. of operations.	
() [] . ->	Highest	Expression. evaluation
- ~ ! & * ++ --		Unary. operations.
* / %.		Multiplicative operators.
+ -		Additive. operators
<< >>		Shifting. operators
< , <=, >, >=		Comp. —"—
= !=		—"— —.
&		
^		
|		
&&		
||		
?: ——————		Conditional execution
=, &=, |=, ^=, += -= *= /= %= >>= <<=		→ Assignment
	Lowest	→ Sequential execution