CodeGuide

QUICK NOTES
FOR SMART LEARNING

EMBEDDED C

QUICK NOTES
FOR SMART LEARNING

SIMPLE EXPLANATIONS | EMBEDDED CONCEPTS | EASY TO LEARN | PRACTICAL APPROACH | EXAM READY

- BASICS OF C
- CONTROL STATEMENTS
- FUNCTIONS
- STRINGS
- POINTERS
- ARRAYS
- REGISTER ACCESS
- AND MORE...



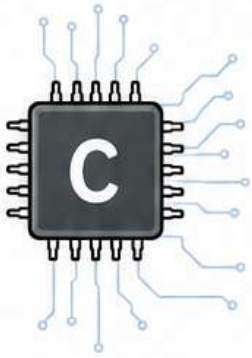
LEARN
PRACTICE
EXCEL

UNDERSTAND • PRACTICE • IMPLEMENT

EMBEDDED C BASIC

What is Embedded C ?

- Embedded C is a subset of C language.
- Used to program microcontrollers and embedded systems.
- Provides direct access to hardware through registers.
- Designed for efficiency, reliability and low resource usage.



Why Embedded C ?

- Fast execution
- Low memory usage
- Direct hardware control
- Reliable and stable
- Portable across platforms



Basic Structure of Embedded C Program

```c
#include <header_file.h>

int main(void)
{
    // Initialization

    while(1)
    {
        // Main Code
    }
    return 0;
}
```

- `#include <header_file.h>` → Header files (Library)
- `int main(void)` → Main function (Entry point)
- `// Initialization` → Initialization section
- `while(1)` → Infinite loop (Program runs forever)
- `return 0;` → Return statement (Not always required)

Key Features

- ★ Hardware oriented
- ★ Event driven
- ★ Real-time execution
- ★ Resource constrained
- ★ Deterministic behavior

Common Data Types

Type	Size (Typical)	Description
char	1 byte	Character / Small integer
int	2 / 4 bytes	Integer
float	4 bytes	Floating point
double	8 bytes	Double precision
void	—	No value type

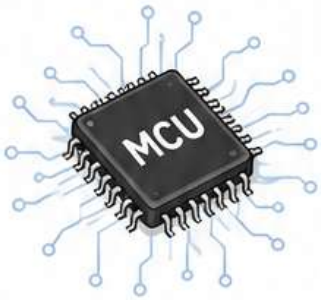
Note: Actual size depends on compiler and architecture.

Embedded C vs General C

Embedded C	General C
Hardware focused	Application focused
Uses registers, I/O	Uses standard libraries
Limited memory	Large memory
Real-time execution	Not real-time
Minimal runtime	Full runtime support

Important Points

- ✓ Program starts from main() function.
- ✓ while(1) keeps the program running.
- ✓ Direct interaction with hardware.
- ✓ No GUI, works in background.
- ✓ Efficiency and stability are priority.



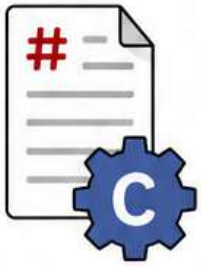
Example Applications

- Microcontroller firmware
- IoT devices
- Motor control systems
- Automotive ECUs
- Medical devices
- Industrial automation

PREPROCESSOR DIRECTIVES

What is Preprocessor?

- Preprocessor directives are instructions given to the preprocessor before compilation.
- They start with # symbol.
- Used for code inclusion, macros, conditional compilation and more.



Common Preprocessor Directives

Directive	Purpose
#include	Includes header files
#define	Defines a macro or constant
#undef	Undefines a macro
#ifdef	Checks if macro is defined
#ifndef	Checks if macro is not defined
#if	Conditional compilation
#endif	Ends conditional block

#include

- Includes header file in the program.
- Two types:
  1. <file.h> → System header file
  2. "file.h" → User defined header file

Example:

```
#include <stdio.h>       // system header
#include "myheader.h"    // user header
```

#define

- Defines a macro or constant.
- Replaced by the value before compilation.
- No memory is allocated.

Example:

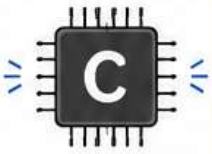
```
#define PI 3.1416
#define MAX 100
int r = 5;
float area = PI * r * r;  // macro PI used
```

Conditional Compilation

- Used to compile code based on conditions.
- Helpful for platform specific code.

```
#define STM32

#ifdef STM32
   // Code for STM32
#else
   // Code for other MCU
#endif
```



Other Directives

```
#undef   → Undefines a macro
#ifndef  → Checks if macro is not defined
#if      → Evaluates a constant expression
#endif   → Ends the conditional block
```

Example Program

```
#include <stdio.h>
#define VERSION  1.0
int main()
{
    printf("Program Version: %.1f\n", VERSION);
    return 0;
}
```

Output

```
Program Version: 1.0
```

Key Points

- ✓ Preprocessor works before compilation.
- ✓ Improves code reusability.
- ✓ No memory used for macros.
- ✓ Helps in platform independent development.

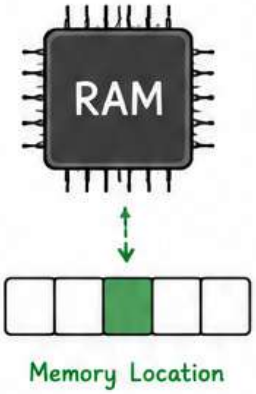




VARIABLES

What is a Variable?

- A variable is a named memory location.
- Used to store data.
- Value can change during program execution.
- Essential for calculations and control.



Declaration Syntax

```
data_type  variable_name  =  value;
```

Example:

```
int   count  =  10;
float  temp  =  25.5;
```

Rules for Naming

- Start with a letter or _
- No spaces allowed
- Case Sensitive
- No special characters except _
- Meaningful names preferred

Valid	Invalid
count	1count
_temp	temp value
speed1	@speed
Max_Value	max-value

Common Data Types

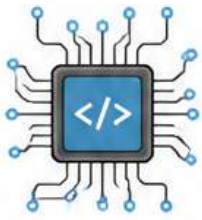
Type	Size (Bytes)	Description
int	2 or 4	Integer value
char	1	Character value
float	4	Floating point value
double	8	Double precision floating point
unsigned int	2 or 4	Positive integer

Example Declarations

```
int      count = 0;            // integer variable
char     grade = 'A';          // character variable
float    temperature = 28.5;   // float variable
unsigned int  distance = 100;  // unsigned int
```

Why Variables are Important?

- Store input from sensors
- Control hardware operations
- Hold intermediate results
- Maintain system status
- Manage counters and flags

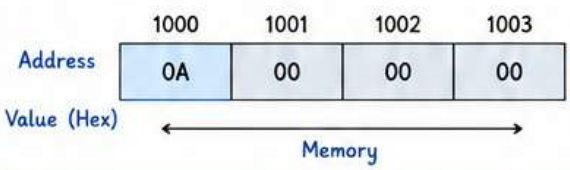


Example in Embedded System

```
int   led_status = 1;     // 1 = ON, 0 = OFF
int   adc_value = 512;    // ADC reading
float voltage = 3.3;      // Voltage value
char  mode = 'A';         // System mode
```

Memory Representation

```
int count = 10;
```



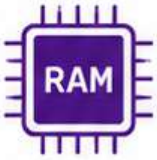
Key Points

- Variables occupy memory.
- Each variable has a data type.
- Value can change during program.
- Use meaningful names for better code.
- Proper use of variables makes code efficient.

Good variable name makes code easy to understand!

Q1. What is a Variable?

```
int count = 10;
```



Answer: A variable is a named memory location used to store data that can change during program execution.

Q2. How to declare a Variable?

```
float temp;
char grade;
int speed;
```



Answer: Specify data type followed by variable name.

Q3. How to initialize a Variable?

```
int age = 20;
float pi = 3.14;
char ch = 'A';
```



Answer: Assign a value to the variable at the time of declaration.

Q4. What are the rules for naming Variables?

- ✓ Start with letter or _
- ✓ No spaces allowed
- ✓ Case sensitive
- ✓ No special characters except _
- ✓ Should be meaningful



Q5. Can we declare multiple variables in one line?

```
int a = 1, b = 2, c = 3;
```



Answer: Yes, we can declare multiple variables of same type in one line.

Q6. What is the default value of a local variable?

```
int x;   // local variable
         // Garbage value
```



Answer: No default value. It contains garbage value.

Q7. What is the default value of a global variable?

```
int x;   // global variable
         // Default value = 0
```



Answer: Default value is 0.

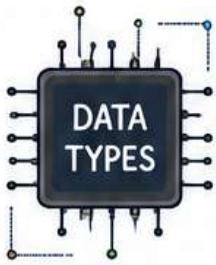
Q8. Difference between local and global variable?

Local Variable	Global Variable
• Declared inside function	• Declared outside function
• Scope: Inside function	• Scope: Whole program
• No default value	• Default value = 0
• Stored in stack	• Stored in data memory

DATA TYPES

WHAT ARE DATA TYPES?

- Data types define the kind of data a variable can store.
- They determine memory size and the range of values.
- C provides different data types for different needs.



CHARACTERISTICS

- ✓ Each type uses a specific amount of memory.
- ✓ Different types can store different ranges.
- ✓ Choose the right type for memory efficiency.

COMMON DATA TYPES

char — Stores a single character — 'A'

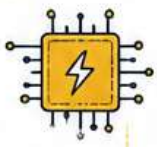
int — Stores whole numbers — 123

float — Stores decimal numbers — 3.14

double — Stores more precise decimals — 3.14159

WHY USE SPECIFIC DATA TYPES?

- ✓ Saves memory
- ✓ Improves performance
- ✓ Ensures portability across devices
- ✓ Avoids overflow issues



KEY POINTS

- ✓ Different data types store different kinds of data.
- ✓ Size of data types may vary in different systems.
- ✓ Use appropriate data types for better performance.
- ✓ Fixed width types are preferred in embedded systems.

BASIC DATA TYPES

Type	Size	Range (Approx.)	Example
char	1 Byte	-128 to 127	'A'
int	2 or 4 Bytes	-32768 to 32767 (or more)	123
float	4 Bytes	±3.4E-38 to ±3.4E+38	3.14
double	8 Bytes	±1.7E-308 to ±1.7E+308	3.14159

FIXED WIDTH INTEGER TYPES

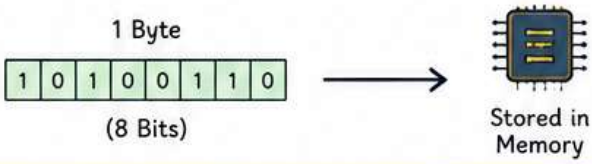
Type	Size	Range
int8_t	1 Byte	-128 to 127
uint8_t	1 Byte	0 to 255
int16_t	2 Bytes	-32768 to 32767
uint16_t	2 Bytes	0 to 65535
int32_t	4 Bytes	-2147483648 to 2147483647
uint32_t	4 Bytes	0 to 4294967295

(From <stdint.h>)

EXAMPLE

```
int   age  = 25;      // Integer
float temp = 36.5;    // Floating Point
char  grade = 'A';    // Character
double pi  = 3.14159; // Double
```

MEMORY REMINDER





Choosing the right data type makes your embedded program more efficient and reliable.

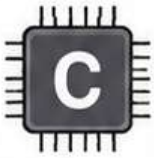


☆ Q1. What are the basic data types in C?

`char` `int` `float` `double` `void`

Example:

```
char ch = 'A';
int num = 10;
float f = 3.14;
double d = 9.81;
```



☆ Q2. What is the size of basic data types? (Typical - 32 bit MCU)

Type	Size
char	1 byte
int	4 bytes
float	4 bytes
double	8 bytes



☆ Q3. What are unsigned data types?

`unsigned char` `unsigned int`
`unsigned long` `unsigned long long`

Example:

```
unsigned int count = 100;
unsigned char led = 1;
```



☆ Q4. What are signed data types?

`signed char` `signed int`
`signed long` `signed long long`

Example:

```
signed int temp = -25;
signed char val = -1;
```



☆ Q5. What are short and long integer types?

`short int` `short unsigned int`
`long int` `long unsigned int`

Example:

```
short int s = 1000;
long int l = 100000;
```



☆ Q6. What is the use of float and double?

`float` → single precision (4 bytes)
`double` → double precision (8 bytes)

Example:

```
float f = 3.14f;
double d = 3.1415926535;
```



☆ Q7. What is the range of int (32 bit system)?

`int range = -2^31 to 2^31 - 1`
`(-2147483648 to 2147483647)`

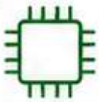


☆ Q8. What are fixed width data types (used in embedded C)?

`int8_t` `uint8_t` `int16_t` `uint16_t`
`int32_t` `uint32_t` `int64_t` `uint64_t`

Example:

```
uint8_t data = 0xFF;
```



☆ Q9. What is the use of void type?

`void` → no value / no return type
Used in functions and pointers.

Example:

```
void func();
void *ptr;
```



☆ Q10. Which data type should be used for hardware registers?

`uint32_t` (32 bit unsigned int)

Example:

```
volatile uint32_t *GPIOA_ODR =
 (uint32_t *)0x40020014;
```

