Quick C

Variable and Data Container

Variable declaration

```
DataType VariableName = Value;
```

Data type and size

- **Character** data type
  - char(1B)
- **Integer** data type
  - int(2B/4B)
- Modifiler
  - signed
  - unsigned
  - short (2B)
  - long(4B)
  - long long(8B)

- integer type Aliases
  - integer type with width of exactly 8, 16, 32 and 64 bits respectively
    - int8_t / uint8_t
    - int16_t / uint16_t
    - int32_t/ uint32_t
    - int64_t/ uint64_t
  - fastest integer type with width of at least 8, 16, 32 and 64 bits respectively
    - int_fast8_t / uint_fast8_t
    - etc.

Data type and size

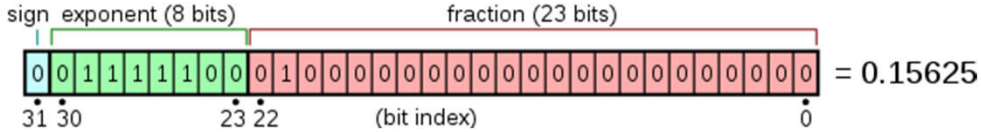
- integer type Aliases
  - **smallest** integer type with width of at least 8, 16, 32 and 64 bits respectively
    - int_least8_t / uint_least8_t
  - integer type capable of holding a pointer to
    - commonly use to perform operation on an address (we can't use some operation such as bitwise on pointer directly)
    - intptr_t /**uintptr_t**
    - etc.

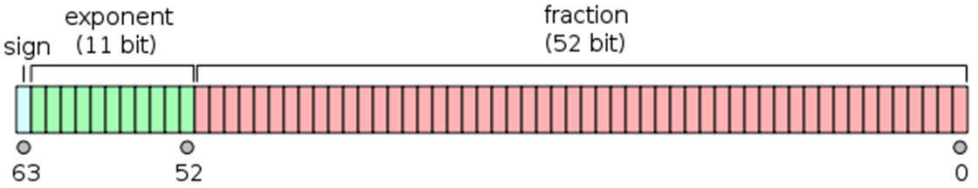
Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
`short`	`short int`	at least **16**	**16**	**16**	**16**	**16**
`short int`						
`signed short`						
`signed short int`						
`unsigned short`	`unsigned short int`					
`unsigned short int`						
`int`	`int`	at least **16**	**16**	**32**	**32**	**32**
`signed`						
`signed int`						
`unsigned`	`unsigned int`					
`unsigned int`						
`long`	`long int`	at least **32**	**32**	**32**	**32**	**64**
`long int`						
`signed long`						
`signed long int`						
`unsigned long`	`unsigned long int`					
`unsigned long int`						
`long long`	`long long int` (C++11)	at least **64**	**64**	**64**	**64**	**64**
`long long int`						
`signed long long`						
`signed long long int`						
`unsigned long long`	`unsigned long long int` (C++11)					
`unsigned long long int`						

Data type and size

$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

- **floating-point** data type
  - float (32b) (~7.2 Digits)
  - double (64b) (~15.9 Digits)
  - long double (128b) (~34.0 Digits)





Data type and size

- **Pointer** data type
  - int*
  - float*
  - void*
  - uint32_t* ptr;

- Operator
  - Reference &a
    - get **memory address** form **variable**
    - uint8_t* ptr = &a ;
  - Dereference *a
    - get/set **value** form **pointer**
    - *ptr = 300 ;

Data type and size

- data type **void**
  - no type ➔ Pointer with no specific type
    - void* ptr;
  - no value ➔ function with no return value
    - void toggleLed()
  - no parameter ➔ function wth no parameter
    - int GetDate(void)

Type	Size in bits	Format	Value range	
			Approximate	Exact
character	8	signed		**-128** to **127**
		unsigned		**0** to **255**
	16	UTF-16		**0** to **65535**
	32	UTF-32		**0** to **1114111** (**0x10ffff**)
integer	16	signed	**± 3.27 · 10^4**	**-32768** to **32767**
		unsigned	**0** to **6.55 · 10^4**	**0** to **65535**
	32	signed	**± 2.14 · 10^9**	**-2,147,483,648** to **2,147,483,647**
		unsigned	**0** to **4.29 · 10^9**	**0** to **4,294,967,295**
	64	signed	**± 9.22 · 10^{18}**	**-9,223,372,036,854,775,808** to **9,223,372,036,854,775,807**
		unsigned	**0** to **1.84 · 10^{19}**	**0** to **18,446,744,073,709,551,615**

Type	Size in bits	Format	Value range	
			Approximate	Exact
binary floating point	32	IEEE-754	•min subnormal: **± 1.401,298,4 · 10^{-45}** •min normal: **± 1.175,494,3 · 10^{-38}** •max: **± 3.402,823,4 · 10^{38}**	•min subnormal: **±0x1p-149** •min normal: **±0x1p-126** •max: **±0x1.fffffep+127**
	64	IEEE-754	•min subnormal: **± 4.940,656,458,412 · 10^{-324}** •min normal: **± 2.225,073,858,507,201,4 · 10^{-308}** •max: **± 1.797,693,134,862,315,7 · 10^{308}**	•min subnormal: **±0x1p-1074** •min normal: **±0x1p-1022** •max: **±0x1.fffffffffffffp+1023**
	80	x86	•min subnormal: **± 3.645,199,531,882,474,602,528 · 10^{-4951}** •min normal: **± 3.362,103,143,112,093,506,263 · 10^{-4932}** •max: **± 1.189,731,495,357,231,765,021 · 10^{4932}**	•min subnormal: **±0x1p-16446** •min normal: **±0x1p-16382** •max: **±0x1.ffffffffffffffep+16383**
	128	IEEE-754	•min subnormal: **± 6.475,175,119,438,025,110,924, 438,958,227,646,552,5 · 10^{-4966}** •min normal: **± 3.362,103,143,112,093,506,262, 677,817,321,752,602,6 · 10^{-4932}** •max: **± 1.189,731,495,357,231,765,085, 759,326,628,007,016,2 · 10^{4932}**	•min subnormal: **±0x1p-16494** •min normal: **±0x1p-16382** •max: **±0x1.ffffffffffffffffffffffffffffp+16383**

Array

- declare ➔DataType ArrayName [ArraySize] = {data};

- access ➔ ArrayName [Position] , *(ArrayName+Position)

- ArrayName ➔ Pointer to Array
- &ArrayName[0] / &ArrayName ➔ Pointer to first element of Array

- sizeof(ArrayName) ➔ size of whole array
- sizeof(&ArrayName) ➔ size of array element

enum– Enumeration
– assign name integral constant

```
enum EnumTag
{
        member1,
        member2,
        member3,….
        …,
        memberN
} enumVar;
```

```
enum DaysOfWeek
{
        Sun,
        Mon,
        Tue,
        Wed,
        Thu,
        Fri,
        Sat
}Day;
```

enum – Enumeration
– assign name integral constant

```
enum DaysOfWeek
{
        Sun,Mon,Tue,Wed,Thu,Fri,Sat
};

enum DaysOfWeek Day;


Day = Mon;

int WorkHours[7];
WorkHours[Mon] = 8;
```

Sun = 0,Mon =1,Tue =2,…

declare variable with enum ,act like declare int

Day = 1

enum

```
enum DaysOfWeek
{
        Sun=5,Mon,
        Tue,Wed,
        Thu=20,Fri,Sat
};
```

Sun = 5,Mon =6,Tue =7,Wed =8,
Thu =20,Fri = 21,Sat =22

```
Day = Mon;
```

Day = 6

enum

```
enum StateOfMachine
        {
                IDLE,
                INIT,
                WORKING,
                CLEANING,
                FAIL
        }
```

```
enum {LO ,HI};


enum{LOW,MID=128,HI=255};
```

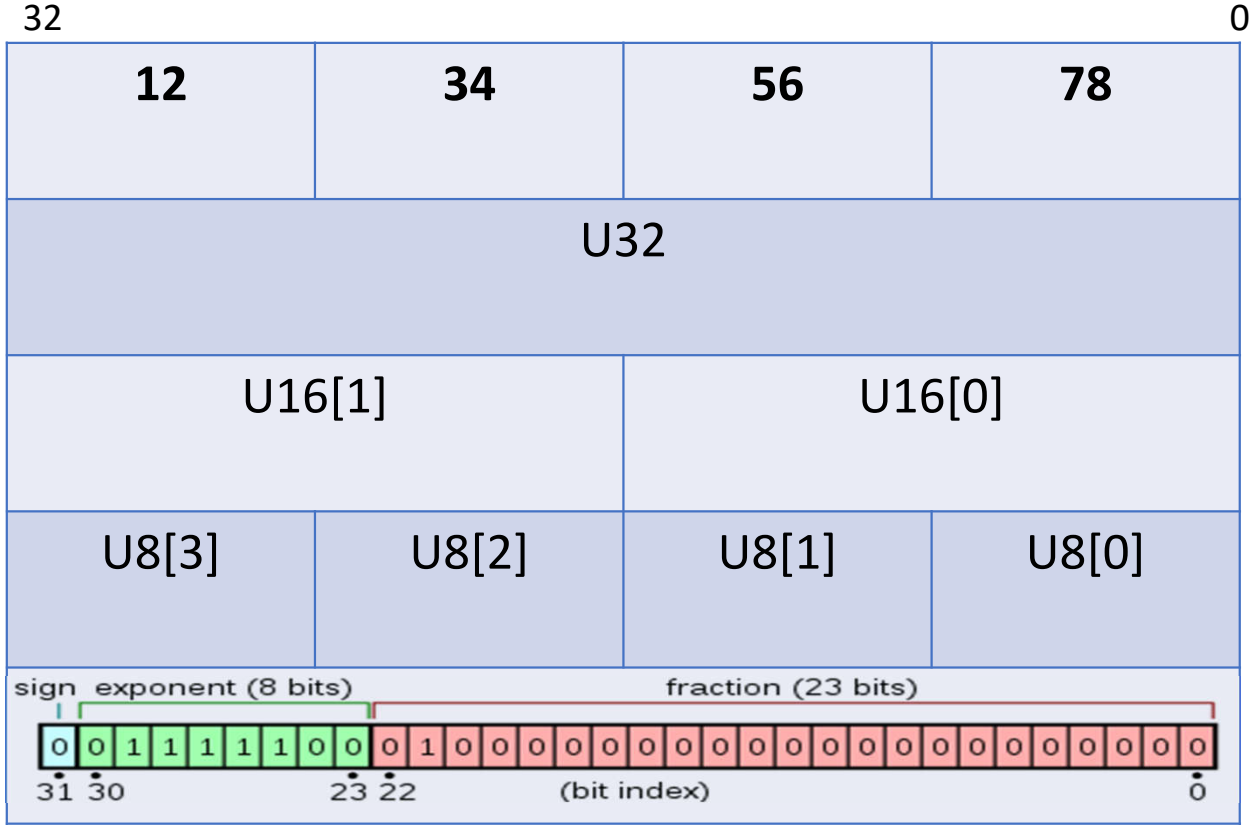
Union
-Share datatype in same memory location

```
union UnionTag
{
        Datatype Member1;
        Datatype Member2;
        Datatype Member3;
        …
         Datatype MemberN;
} UnionVar;
```

```
union U32BitsConv
{
        uint32_t U32;
        uint16_t U16[2];
        uint8_t U8[4];
        float   F32;
} U32Convert;
```

Union
-Share datatype in same memory location

```
union U32BitsConv
{
        uint32_t U32;
        uint16_t U16[2];
        uint8_t U8[4];
        float    F32;
} U32Convert;

U32Convert.U32 = 0x12345678
```



Structure - declare

```
struct StructTag{
        int a;
        float b;
        char c[3];
} structName = { 123 , 3.14 , {‘P’, ‘u’, ‘n’} } ;
```

```
struct StructTag
{
        int a;
        float b;
        char c[3];
};

struct StructTag structName = { 123 , 3.14 , {‘P’, ‘u’, ‘n’} } ;
struct StructTag struct2;

struct2 = structName ; //Copy struct
```

Structure access

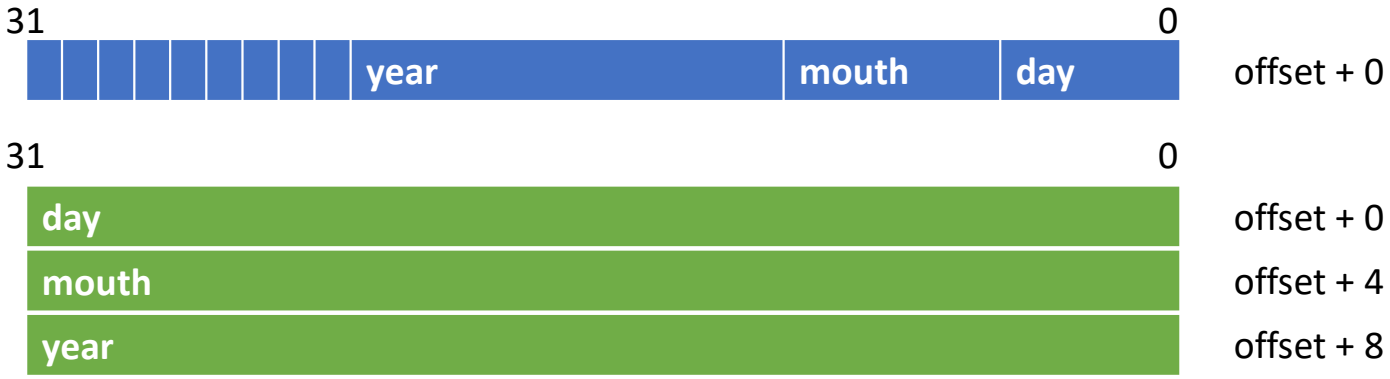
```
struct StructTag{
        int a;
        float b;
        char c[3];
} structName = { 123 , 3.14 , {'P', 'u', 'n'} } ;

struct StructTag* ptrToStruct = &structName
```

```
structName.a = 456;
float EstPi = structName.b;
structName.c[1] = 2

ptrToStruct->a =456;
(*ptrToStruct).a =456;
```

structure - bitfield

```
struct Date                                    size : 4 Byte
{
        unsigned int day : 5;
        unsigned int mouth : 4;
        unsigned int year :12;
} stDate = {31,12,2021} ;
```

```
struct Date                                    size : 12Byte
{
        unsigned int day;
        unsigned int mouth;
        unsigned int year;
} stDate = {31,12,2021} ;
```



structure - bitfield

```
struct DeviceStatus                                    size : 4 Byte
{
        uint32_t RunningFlag:1;
        uint32_t FailFlag :1;
        uint32_t YoloFlag :1;
        uint32_t RunningFlag2:1;
        uint32_t FailFlag2 :1;
        uint32_t YoloFlag2 :1;
        uint32_t RunningFlag3:1;
        uint32_t FailFlag3 :1;
        uint32_t YoloFlag3 :1;
} DVS = {0} ;
```

```
struct DeviceStatus                                    size : 2Byte
{
        uint8_t RunningFlag:1;
        uint8_t FailFlag :1;
        uint8_t YoloFlag :1;
        uint8_t RunningFlag2:1;
        uint8_t FailFlag2 :1;
        uint8_t YoloFlag2 :1;
        uint8_t RunningFlag3:1;
        uint8_t FailFlag3 :1;
        uint8_t YoloFlag3 :1;

} DVS = {0} ;
```

typedef - give Type a new Name

➔typedef typeName NewTypeName;

typedef uint8_t Byte;
Byte Var = 255;

typedef unsigned char uint8_t;

typedef uint8_t MachineState;
MachineState State=0;

```
struct Date
{
        unsigned int day : 5;
        unsigned int mouth : 4;
        unsigned int year :12;
} ;

struct Date stDate = {31,12,2021} ;
```

```
typedef struct Date
{
        unsigned int day : 5;
        unsigned int mouth : 4;
        unsigned int year :12;
}DateStructure ;

DateStructure stDate = {31,12,2021} ;
```

```
union U32BitsConv
{
        uint32_t U32;
        uint16_t U16[2];
        uint8_t U8[4];
        float    F32;
} ;

union U32BitsConv U32convert;
```

```
typedef union U32BitsConv
{
        uint32_t U32;
        uint16_t U16[2];
        uint8_t U8[4];
        float    F32;
} U32Convertor;

U32Convertor U32convert;
```

```
enum DaysOfWeek
{
        Sun,Mon,Tue,Wed,Thu,Fri,Sat
};

enum DaysOfWeek Day;
```

```
typedef enum __DaysOfWeek
{
        Sun,Mon,Tue,Wed,Thu,Fri,Sat
}DayOfWeek;

DaysOfWeek Day;
```

Bits Modifier

REGISTER 5-1: WRITE COMMAND REGISTER FOR MCP4922 (12-BIT DAC)

W-x	W-x	W-x	W-0	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
$\overline{A/B}$	BUF	$\overline{GA}$	$\overline{SHDN}$	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
bit 15															bit 0

bit 15 **$\overline{A/B}$:** DAC_A or DAC_B Selection bit
- 1 = Write to DAC_B
- 0 = Write to DAC_A

bit 14 **BUF:** V_{REF} Input Buffer Control bit
- 1 = Buffered
- 0 = Unbuffered

bit 13 **$\overline{GA}$:** Output Gain Selection bit
- 1 = 1x ($V_{OUT} = V_{REF} * D/4096$)
- 0 = 2x ($V_{OUT} = 2 * V_{REF} * D/4096$)

bit 12 **$\overline{SHDN}$:** Output Shutdown Control bit
- 1 = Active mode operation. VOUT is available.
- 0 = Shutdown the selected DAC channel. Analog output is not available at the channel that was shut down. VOUT pin is connected to 500 kΩ (typical).

bit 11-0 **D11:D0:** DAC Input Data bits. Bit x is ignored.

Bits Modifier

REGISTER 5-1: WRITE COMMAND REGISTER FOR MCP4922 (12-BIT DAC)

W-x	W-x	W-x	W-0	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
$\overline{A/B}$	BUF	$\overline{GA}$	$\overline{SHDN}$	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
bit 15															bit 0

```
typedef union _MCP4922_To_U16
{
	struct MCP4922
	{
		uint16_t data :12;  //LSB
		enum _SHDN{SHUTDOWN,ACTIVE} SHDN :1;
		enum _GAIN{X1,X2} GA  :1;
		enum _BUFF{UNBUFF,BUFF} buf :1;
		enum _AB{A,B}AB:1; //MSB
	} MCP;
	uint16_t U16;
}MCP4922_To_U16;
```

bit 15 **$\overline{A/B}$:** DAC_A or DAC_B Selection bit
1 = Write to DAC_B
0 = Write to DAC_A

bit 14 **BUF:** V_{REF} Input Buffer Control bit
1 = Buffered
0 = Unbuffered

bit 13 **$\overline{GA}$:** Output Gain Selection bit
1 = 1x ($V_{OUT} = V_{REF} * D/4096$)
0 = 2x ($V_{OUT} = 2 * V_{REF} * D/4096$)

bit 12 **$\overline{SHDN}$:** Output Shutdown Control bit
1 = Active mode operation. VOUT is available.
0 = Shutdown the selected DAC channel. Analog output is not available at the channel that was shut down. VOUT pin is connected to 500 kΩ (typical).

bit 11-0 **D11:D0:** DAC Input Data bits. Bit x is ignored.

```
MCP4922_To_U16 DATA;

DATA.MCP.data = 0xAA5;
DATA.MCP.SHDN = ACTIVE;
DATA.MCP.buf = UNBUFF;
DATA.MCP.GA =X1;
DATA.MCP.AB =A;

DATA.U16; //use to Send SPI
```

bit 15 **$\overline{A/B}$: DAC_A or DAC_B Selection bit**
1 = Write to DAC_B
0 = Write to DAC_A

bit 14 **BUF: V_{REF} Input Buffer Control bit**
1 = Buffered
0 = Unbuffered

bit 13 **$\overline{GA}$: Output Gain Selection bit**
1 = 1x ($V_{OUT} = V_{REF} * D/4096$)
0 = 2x ($V_{OUT} = 2 * V_{REF} * D/4096$)

bit 12 **$\overline{SHDN}$: Output Shutdown Control bit**
1 = Active mode operation. VOUT is available.
0 = Shutdown the selected DAC channel. Analog output is not available at the channel that was shut down. VOUT pin is connected to 500 kΩ (typical).

bit 11-0 **D11:D0:** DAC Input Data bits. Bit x is ignored.

DATA	MCP4922_To_U16	{...}
MCP	struct MCP4922	{...}
data	uint16_t	0xaa5
SHDN	enum _SHDN	0x1
GA	enum _GAIN	0x0
buf	enum _BUFF	0x0
AB	enum _AB	0x0
U16	uint16_t	0x1aa5

Variable Storage Class

	Scope	Storage	Life
auto	within block	Stack	End of block
static	within block	Data Section	End of program
extern	Multiple file / Global	Data Section	End of program
register	within block	CPU Register	End of block

auto – Default Storage Class ,and Automatically assign to any variable(that not use static extern or register)

static - preserving value until end of program

extern – This Variable define somewhere else but not this file

Register – If you(compiler) can , assign this variable to CPU Register

static

```
int countUp()
{
        static int a;
        return a++;
}
```

by default , static variable Initial value is 0

A preserved value when end of function

extern

Tell complier that b is declare in another file

```
extern int b;
void countUp()
{
        extern int a;
        return a++;
}
```

Tell complier that a is declare in another file , but only visible by this function

Main.c

```
TIM_HandleTypeDef htim1;
UART_HandleTypeDef huart2;
```

User.cpp

```
extern TIM_HandleTypeDef
htim1;
extern UART_HandleTypeDef
huart2;
```

Register

```
register int i ;
for(i=0;i<10;i++)
{
        //do something
}
```

Compiler will put i in CPU register instead of RAM (If they can)

Register variable can’t access by pointer

Register can’t used in global scope

Used when variable need frequently access and used in small size

Variable type qualifiers – Tell Complier something important (for better optimize)

- const – DO NOT EDIT THIS
- volatile – This can change form other things (like hardware)
- restrict – This pointer is only way to access this value

const – DO NOT EDIT THIS

- Tell compiler that “This variable value will not be change” , make variable can store in Read-only Memory (sometime in size code directly)
- const float Pi = 3.1415;

const int *ptr = &i ptr is pointer to constant “*ptr” **can’t** modified , “ptr” **can** modified “i” can or cannot be constant	int const *ptr = &i ptr is pointer to constant int “*ptr” **can’t** modified , “ptr” **can** modified “i” must be constant int
int *const ptr = &i ptr is constant pointer to int “*ptr” **can** modified , “ptr” **can’t** modified “i” can or cannot be constant	const int *const ptr = &i ptr is constant pointer to constant int “*ptr” **can’t** modifild , “*ptr” **can’t** modified “i” must be constant int

volatile – This can change form other things

- Tell compiler that “this variable can be edit by someone that not in this program at anytime” ➔ compiler not optimize this variable out and try to read every time we call it.

```
GPIOA->BSRR = 0x1;
```

```
70
71 typedef struct
72 {
73   __IO uint32_t CR1;         /*!< TIM control register 1,
74   __IO uint32_t CR2;         /*!< TIM control register 2,
75   __IO uint32_t SMCR;        /*!< TIM slave mode control register,
76   __IO uint32_t DIER;        /*!< TIM DMA/interrupt enable register,
```

```
#define      __IO    volatile
```

restrict – This pointer is only way to access this value

- Tell compiler that “this value from **pointer** will not change by anything except this **pointer**”

```
void test(int* a, int* b, int* restrict c)
{
        *a += *c;
        *b += *c;
}
```

Compiler generated code that not read *c to register every time.

Variable Initialization and constants

Variable should Initial before use to avoid garbage data
(expect static and global that set to 0 by compiler)

- int a = 1;
- int b[]= {1,2,3}
- char c = ‘g’;
- float f = **1.0**;
- char h[] =“Hello World”
- int g[10] ={0}

‘ ’ convert ASCII to Number

.0 make floating point constant
useful in calculation
1 / 3 = 0 Because int / int = int
but 1/3.0 = 0.333333 because int / float = float

“ ” make array of char constant
size is 12 because of white space and Null

initial all element set to 0

Variable Initialization and constants

- uint8_t a = 255U; //Unsigned
- uint16_t b = 254u; //Unsigned
- uint8_t c = 0xFF; //Hex
- uint8_t d = 056; //Oct
- uint8_t e = 0b01001100; //Bin
- int32_t f = -123L; //Long
- int32_t g = 123ul; //Unsigned long
- uint64_t g= -1LL; //Unsigned Long Long
- float h = 2.99792458E8 ; // 2.99792458x10^8

Variable Initialization and constants

```
typedef struct Date
{
        unsigned int day : 5;
        unsigned int mouth : 4;
        unsigned int year :12;
}DateStructure ;

DateStructure stDate = {31,12,2021} ;
```

```
typedef struct Date
{
        unsigned int day : 5;
        unsigned int mouth : 4;
        unsigned int year :12;
}DateStructure ;

DateStructure stDate = {0} ;
```

```
struct StructTag{
        int a;
        float b;
        char c[3];
} structName = { 123 , 3.14 , {'P', 'u', 'n'} } ;
```

```
struct StructTag{
        int a;
        float b;
        char c[3];
} structName = { 0} ;
```

Variable Name

- We should Name Variable properly to show other people (and yourself in next week)What this variable does.
- We have auto complete, don’t scare to name it a little bit longer

Common rule
-contain letters ,digits , Underscore
-begin with letter or underscore
-no whitespace or special character
-not reserved word
-Case sensitive

some common style and tip

- Macro and Constant value ➔ ALL_UPPER_CASE_WITH_UNDERSCORE
- something that shouldn't access by anyone ➔ _something
- function and variable name ➔ wordGroup_wordGroup
- pointer ➔ ptrVariable or p_Variable
- some quick loop ➔ i,j,k are fine ^ ^
- tips: add some common name at start of variable to easy auto correct
  - like HAL_ADC_XXXX

Preprecessor – Do before compile (marco)

- #include ➔ include library can be .h ,.c , .hpp ,etc
- #define ➔ change / define keyword that will replace with constant / expression
  - #define DEBUG
  - #define MAX_VELOCITY 200
  - #define circleArea(r) (3.1415*r*r)

```
#define __HAL_TIM_GET_COMPARE(__HANDLE__, __CHANNEL__) \
  (((__CHANNEL__) == TIM_CHANNEL_1) ? ((__HANDLE__)->Instance->CCR1) :\
   ((__CHANNEL__) == TIM_CHANNEL_2) ? ((__HANDLE__)->Instance->CCR2) :\
   ((__CHANNEL__) == TIM_CHANNEL_3) ? ((__HANDLE__)->Instance->CCR3) :\
   ((__HANDLE__)->Instance->CCR4))
```

Preprecessor – Do before compile (marco)

- #ifdef - #endif ➔ use this code if define
- Exp

```
#define DEBUG

#ifdef DEBUG
        print(someLog);
#endif
```

Preprecessor – Do before compile (marco)

- #ifndef - #endif ➔ use this code if not define
- Exp

```
#define NO_LOG

#ifndef NO_LOG
        print(someLog);
#endif
```

Preprecessor – Do before compile (marco)

- #if condition
- #undef ➔ undefine
- #else

Dynamic memory Allocation

- allocate memory in heap space on runtime
- size of variable can set on runtime
- **can be dangerous** ,with memory leak or stack / heap overflow
- compiler can’t count memory size that we use in Dynamic memory
- function
  - malloc(sizeInByte);
  - calloc(number , sizeInByte);
  - realloc(ptr,newSize);
  - free(ptr)

malloc , calloc

- ptr = (cast-type*) malloc(byte-size) // faster , init with garbage
  - allocate int[n]

```
int* ptr;
ptr = (int*)malloc(n * sizeof(int));
```

ptr = (cast-type*)calloc(n, element-size); //slower ,init with zero

  - allocate int[n]

```
int* ptr;
ptr = (int*) calloc(n , sizeof(int));
```

reallocate

- ptr = realloc(ptr, newSize);

```
int* ptr;
ptr = (int*)malloc(n * sizeof(int));

 realloc(ptr, 20);
```

free()

- compiler don’t manage any space that allocate with malloc / calloc
  - resource stay use even program or function end
  - if we don’t free memory that don’t use any more , garbage will collect ,and boom overflow

  - always use free(ptr) and recheck that every malloc/calloc we use ,we have free

Section – declare variable at specific space

- uint8_t Rx_Buff[128] __attribute__((section(“RAM_D3"))) = {0};
  - assign Rx_Buff to sram4 in stm32h745;

operator

Precedence	Operator	Description	Associativity
1	++ -- () [] . -> (*type*){*list*}	Suffix/postfix increment and decrement Function call Array subscripting Structure and union member access Structure and union member access through pointer Compound literal(C99)	Left-to-right
2	++ -- + - ! ~ (*type*) * & sizeof _Alignof	Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT Cast Indirection (dereference) Address-of Size-of Alignment requirement(C11)	Right-to-left
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-left
14	= += -= *= /= %= <<= >>= &= ^= |=	Simple assignment Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

arithmetic

+a
-a
a + b
a - b
a * b
a / b
a % b (mod)

bitwise

~a	bitwise NOT
a & b	bitwise AND
a | b	bitwise OR
a ^ b	bitwise XOR (Use as bit toggle)
a << b	bitwise ShiftLeft
a >> b	bitwise ShiftRight

comparison

a == b
a != b
a < b
a > b
a <= b
a >= b

logical

!a
a && b
a || b

Note:

logical and comraison always return true(1) or false(0)

Note2:

float a =0.1; //f = 0.100000001490116119384765625
double b =0.1; g = 0.1000000000000000055511151231257827021181583404541015625
(a==b) is false

Note3:

for optimization
A&& B Try A that use to be False
A||B Try A that use to be True

assignment

- a = b
  a += b
  a -= b
  a *= b
  a /= b
  a %= b
  a &= b
  a |= b
  a ^= b
  a <<= b
  a >>= b

increment
decrement

++a

--a

a++

a--

```
i = 1;
j = ++i;

i == 2, j == 2
```

```
i = 1;
j = i++;

i == 2, j == 1
```

Decision

- if elseif else
- switch case
- (C)?T:F

if else – condition base decision

```
if(condition) {
   // Statements inside body of if
}
```

```
if(power < 12) {
   chargebattery();
}
```

```
if(power < 12) chargebattery();
```

Condition in C:	
Zero	= False
non-Zero	= True
0	= False
0.0	= False
-0.0	= False
-1	= True
1	= True
1235689	= True
‘0’	= True

if else

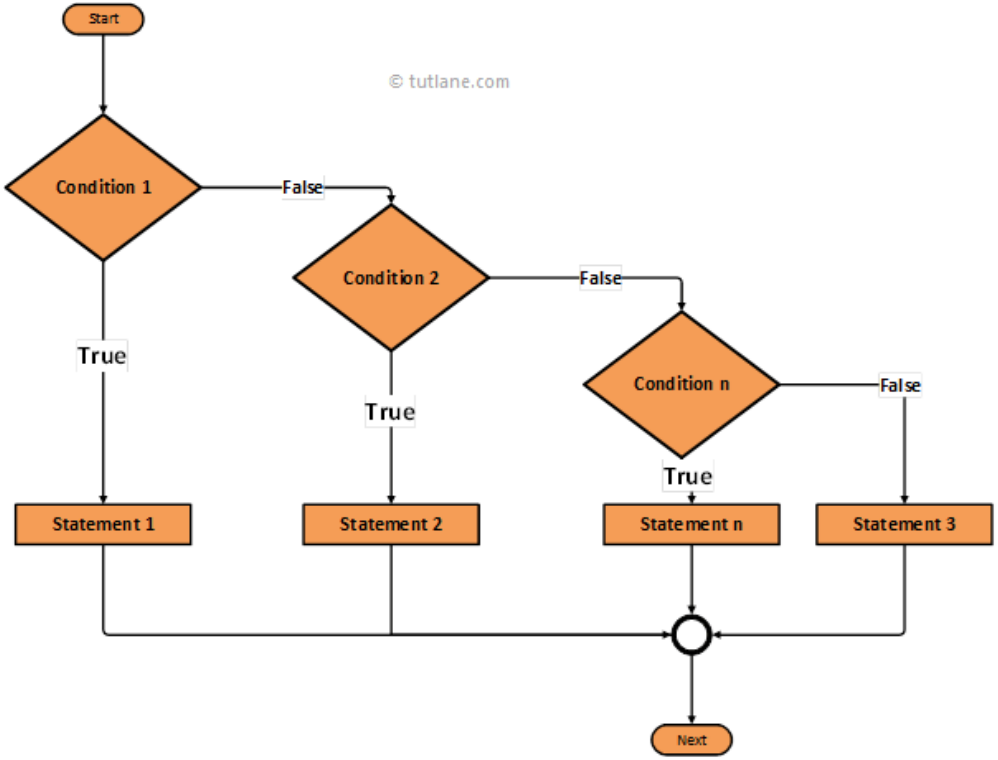
```
if(condition) {
   // Statements inside body of if
}
else
{

}
```

```
if(power < 12) {
   chargebattery();
}
else
{
  working();
}
```

```
if(power < 12) chargebattery();
else Working();
```

if else

```
if(condition) {
   // Statements inside body of if
}
else if(condition)
{


}
else
{


}
```



if optimization

- common condition First
  - if (machineState == working) and “working” have 80% of time

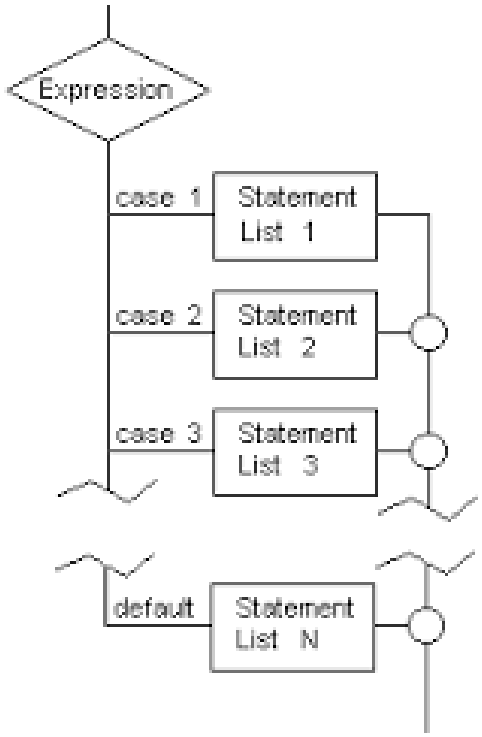
- common condition calculate outside

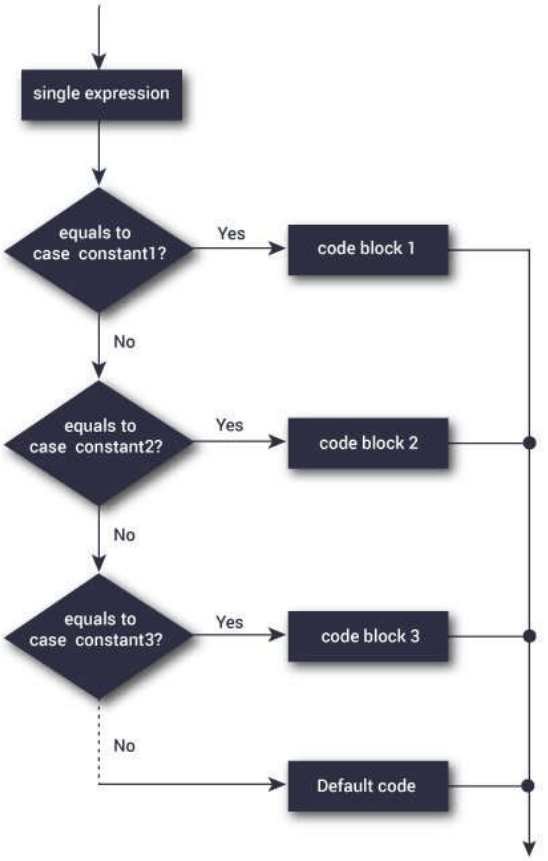
```
if(a==2||b==3&&c==4)
{
}
else if(a==2||b==3&&c==5)
{
}
```

```
int common = (a==2||b==3);
if(common &&c==4)
{
}
else if(common &&c==5)
{
}
```

switch case - value base decision

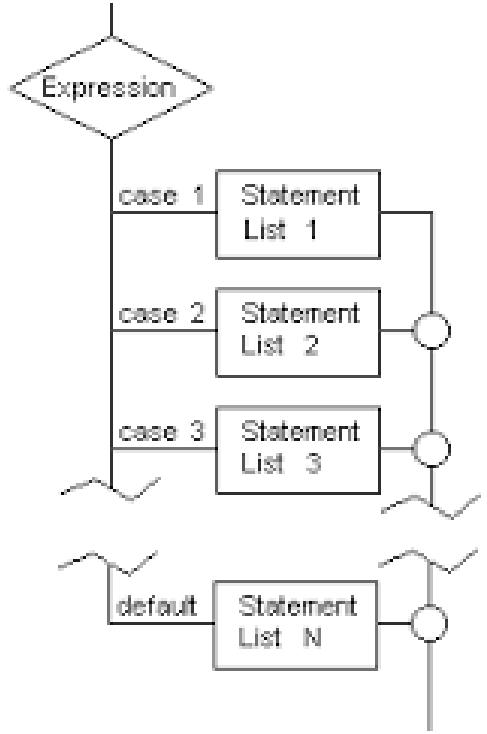
```
switch(Variable)
{
        case 0:
        case 2:
                something1();
        break;
        case 1:
        case 4:
                something2();
        break;
        default:
        break
}
```



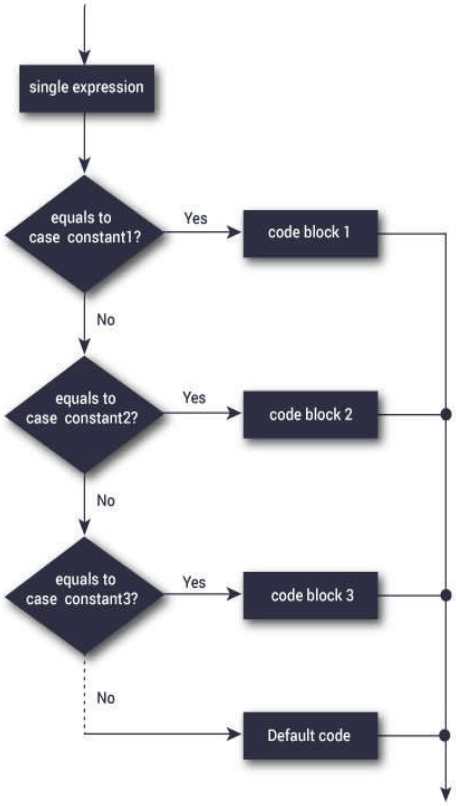


switch case optimize

```
switch(Variable)
{
        case 0:
          something1();
         break;
        case 2:
          something2();
        break;
        case 1:
          something3();
         break;
        case 4:
          something4();
        break;
        default:
        break
}
```




```
switch(Variable)
{
        case 0:
          something1(); break;
        case 10:
          something2(); break;
        case 20:
          something3(); break;
        case 30:
          something4(); break;
        break;
        default:
        break
}
```



switch case optimize

1. use case with consecutive numbers or enum
2. separate common task ,Grouping common task with consecutive numbers

```
switch(Variable)
{
        case 0:
        commonTask()
          something1();
        case 2:
          something2();
        break;
        case 1:
        commonTask()
          something3();
```

```
switch(Variable)
{
        case 0:
        case 1:
        commonTask()
        switch(variable)
        {
                case 0:
                something1(); break;
                case 1:
                something2(); break;
        }
        break;
        case 2:
          something3(); break;
        case 3:
          something4(); break;
        default:
        break
}
```

(C)?T:F - fast condition base decision (that return value)

(condition) ? [true value] : [False value]

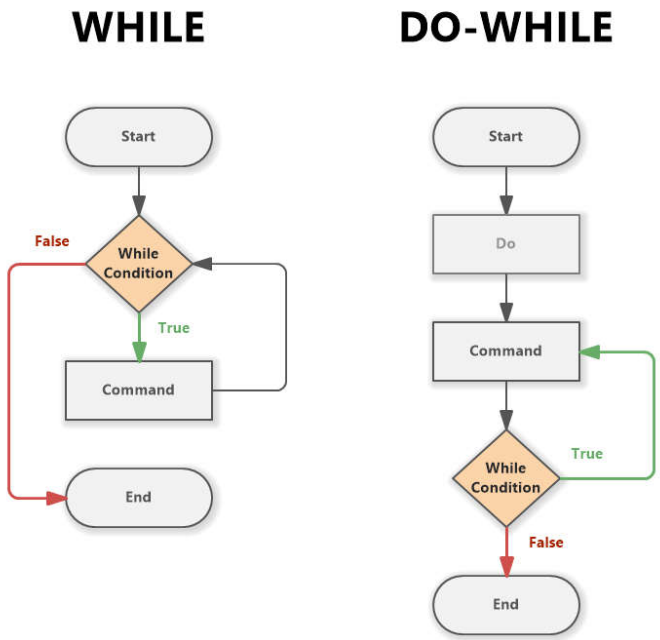
```
int A = 3;
int B = 4;

int C = (A<B)? 5 : 6; // C = 5

int Max = (A>B) ? A :B ; Max = 6
```

Loop

- While /do While
- For

while do while – loop with condition

```
while (condition)
{
    something();
}

while(condition)something;

while(condition);
```




```
do{
    something();
} while (condition)
```

for – loop with counter(and condition)

```
for(int i =0; i<10;i++)
{
        something();
}

for(int i =0; i<10;i++) something();
```

for([init];[condition];[iteration])

[init] ➔ run 1 time on start loop
[condition] ➔ loop stop when false
[iteration] ➔ run every end of loop

for – loop with counter(and condition)

```
char str[128] = "Some Text";
char *p;

for (p = str; *p ; p++)
{
   // Code
}
```

for([init];[condition];[iteration])

[init] ➔ run 1 time on start loop
[condition] ➔ loop stop when false
[iteration] ➔ run every end of loop

loop control

- effect most inner loop
  - break; ➔ stop and exit loop
  - continue; ➔ stop current loop and goto next loop

```
for(int i =0; i<10;i++)
{
        if(i == 5) // i=5 skip loop
        {
                continue;
        }
        something();
}
```

loop optimize

- avoid calculation and pointer dereference ,Keep loop body small
- if loop iteration is so small, DON'T USE LOOP
- try loop by decrease to zero

```
for (int i = 0; i < 1000; ++i) {
      arr[i] = (((c % d) * a / b) % d) * i;
  }
```

```
int temp = (((c % d) * a / b) % d);

  for (int i = 1000; i > 0; --i) {
     arr[i] = temp * i;
  }
```

```
int temp = *iptr;

  for (int i = 1; i < 11; ++i) {
     temp = temp + i;
  }

  *iptr = temp;
```

Loop Forever

- while(1){};
- for(;;){};
  - Doing the same thing, and compiler optimized it anyway

function

```
return_type function_name( parameter list )
{
   //body of the function
 return value;
}
int add(int a,int b)
{
        return a+b;
}
```

function parameter

- Call by value
  - void funcA(int A)
  - A is a copy value of variable , change A don’t change original variable

- Call by Reference
  - void funcB(int* B)
  - B is a copy pointer of variable ,any change in B value effect value at original address

Inline – “just copy this function to the code every time it call”

```
inline float P(float e)
{
    return kp*e
}
force way:
 __attribute__((always_inline)) float P(float e)
{
    return kp*e
}
```

weak

- weak / __weak / __attribute__((weak))
  - complier don’t use this function if it declare somewhere else.

```
__weak void HAL_TIM_ErrorCallback(TIM_HandleTypeDef *htim)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(htim);

  /* NOTE : This function should not be modified, when the callback is needed,
            the HAL_TIM_ErrorCallback could be implemented in the user file
   */
}
```

Common C programming Structure

Documentation	/* *Author : Putthinart *File Name: *License: */
Link section	#include <stdio.h> #include “UserLib.h”
Definition	#define MAX_VAL 200 typedef char Machine state enum ,Union,structure
Global Declaration Function Prototype	const float PI =3.14 void userFunction();
Main	int main() {return 0;}
Sub program	void userFunction() {return;}

Documentation

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
 * All rights reserved.</center></h2>
 *
 * This software component is licensed by ST under BSD 3-Clause license,
 * the "License"; You may not use this file except in compliance with the
 * License. You may obtain a copy of the License at:
 *                        opensource.org/licenses/BSD-3-Clause
 *
 ******************************************************************************
 */
/* USER CODE END Header */
```

```
/*
 * PNGUartDecode.c
 *
 *  Created on: Aug 21, 2021
 *      Author: AlphaP
 */
```

```
/**
  ******************************************************************************
  * @file    stm32h7xx_hal_gpio.c
  * @author  MCD Application Team
  * @brief   GPIO HAL module driver.
  *          This file provides firmware functions to manage the following
  *          functionalities of the General Purpose Input/Output (GPIO) peripheral:
  *           + Initialization and de-initialization functions
  *           + IO operation functions
  *
  @verbatim
  ==============================================================================
                    ##### GPIO Peripheral features #####
  ==============================================================================
  [..]
    (+) Each port bit of the general-purpose I/O (GPIO) ports can be individually
        configured by software in several modes:
        (++) Input mode
        (++) Analog mode
        (++) Output mode
        (++) Alternate function mode
        (++) External interrupt/event lines
```

Link Section

- #include <stdio.h> //search in library path first
- #include “UserCode.h” // search in local path first
- #include “User/UserCode.h” // search in local with sub folder

Library - Grouping and management

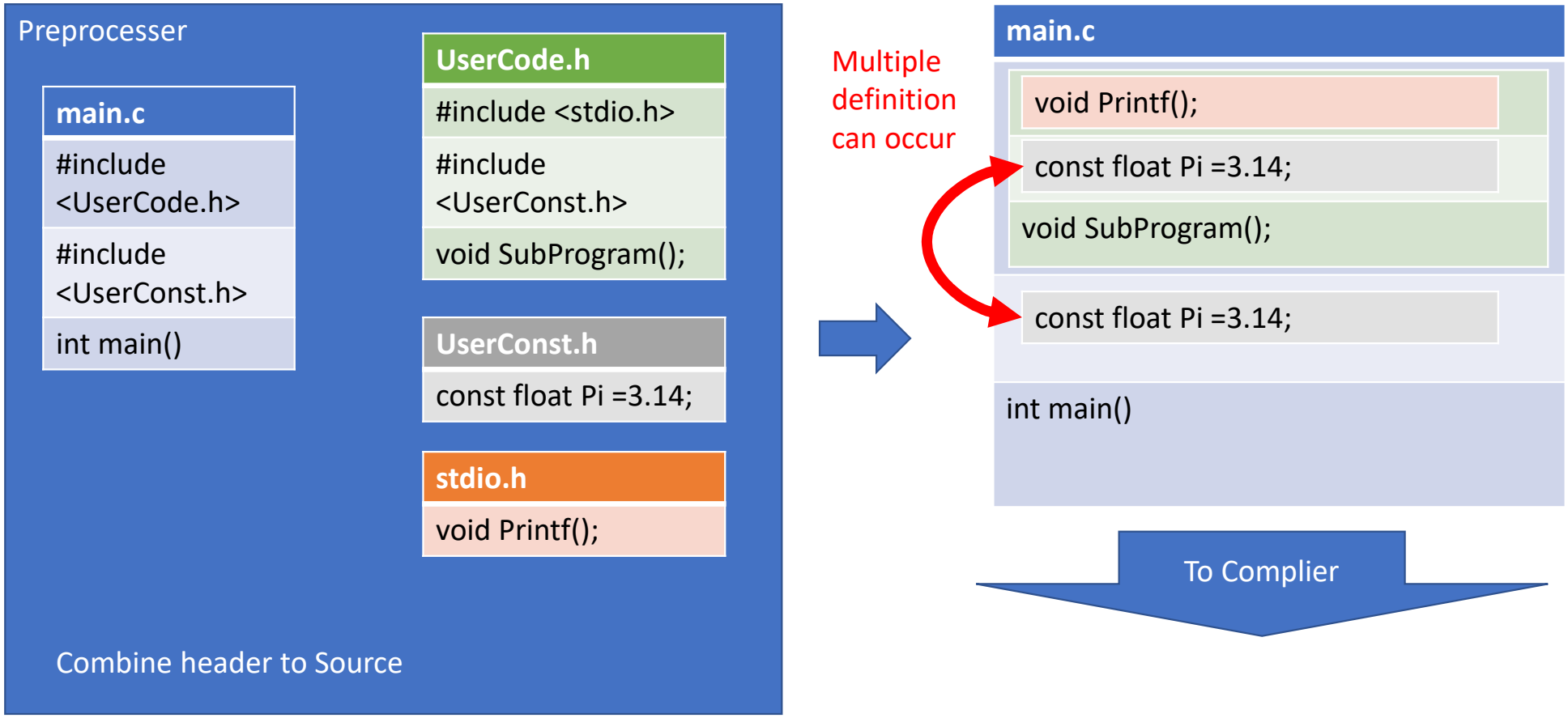
Tell “What” this library have

UserCode.h **Global interface / declaration**
Documentation
Link section
Definition
Global Declaration Function Prototype
sub program*

Tell “How” this library Do

UserCode.c **Private implementation**
Documentation
Link section
Definition
Private Declaration Function Prototype
sub program

Library – Header file - how it work



Library – Header file

UserCode.h
#ifndef __USERCODE_H
#define __USERCODE_H
#include <stdio.h>
#include <UserConst.h>
void SubProgram();
#endif

important

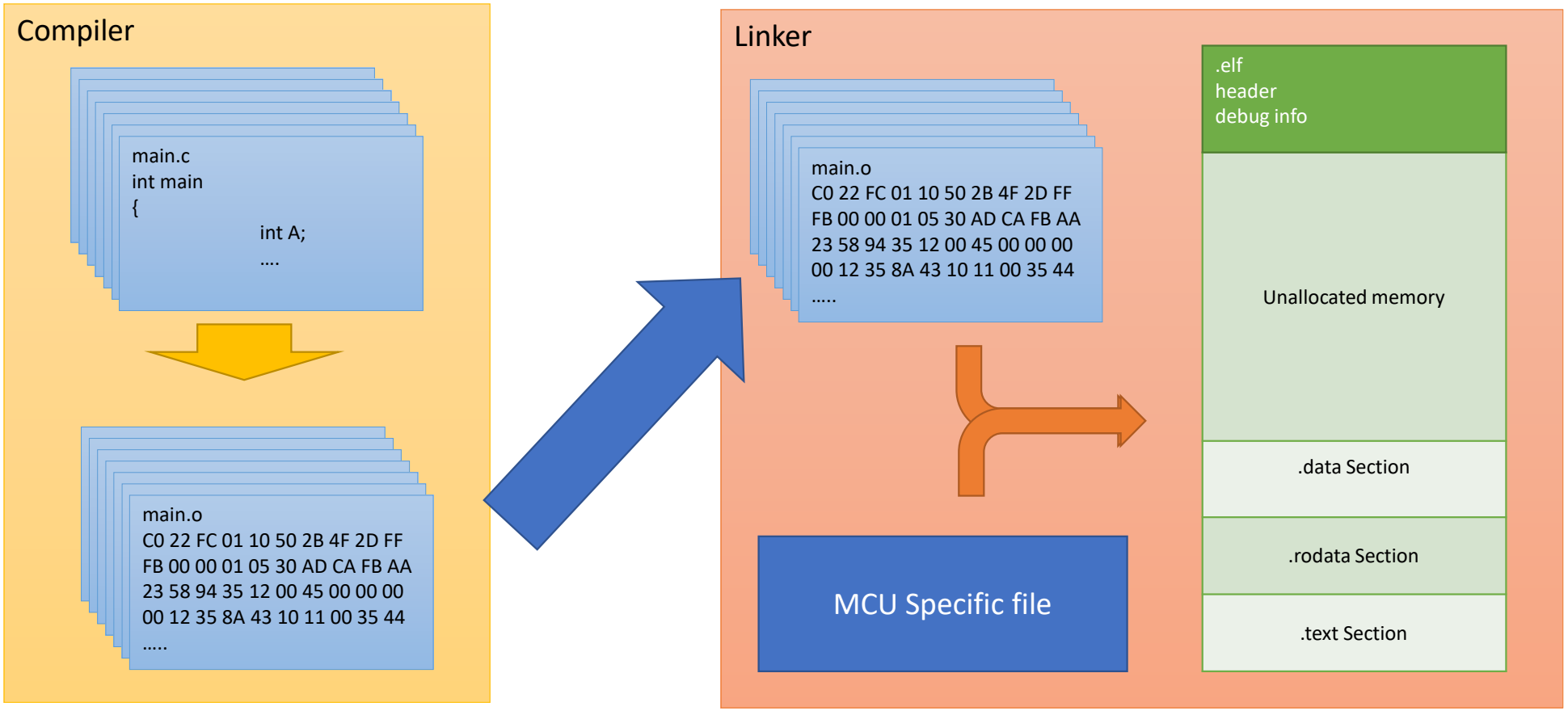
```
/*
 * BMPUartDecode.h
 *
 *  Created on: Aug 21, 2021
 *      Author: AlphaP
 */

#ifndef INC_BMPUARTDECODE_H_
#define INC_BMPUARTDECODE_H_

#include "stm32h7xx_hal.h"

void BMPDecoder(uint8_t dataIn, uint8_t *array);



#endif /* INC_BMPUARTDECODE_H_ */

```

Library – Code file - how it work

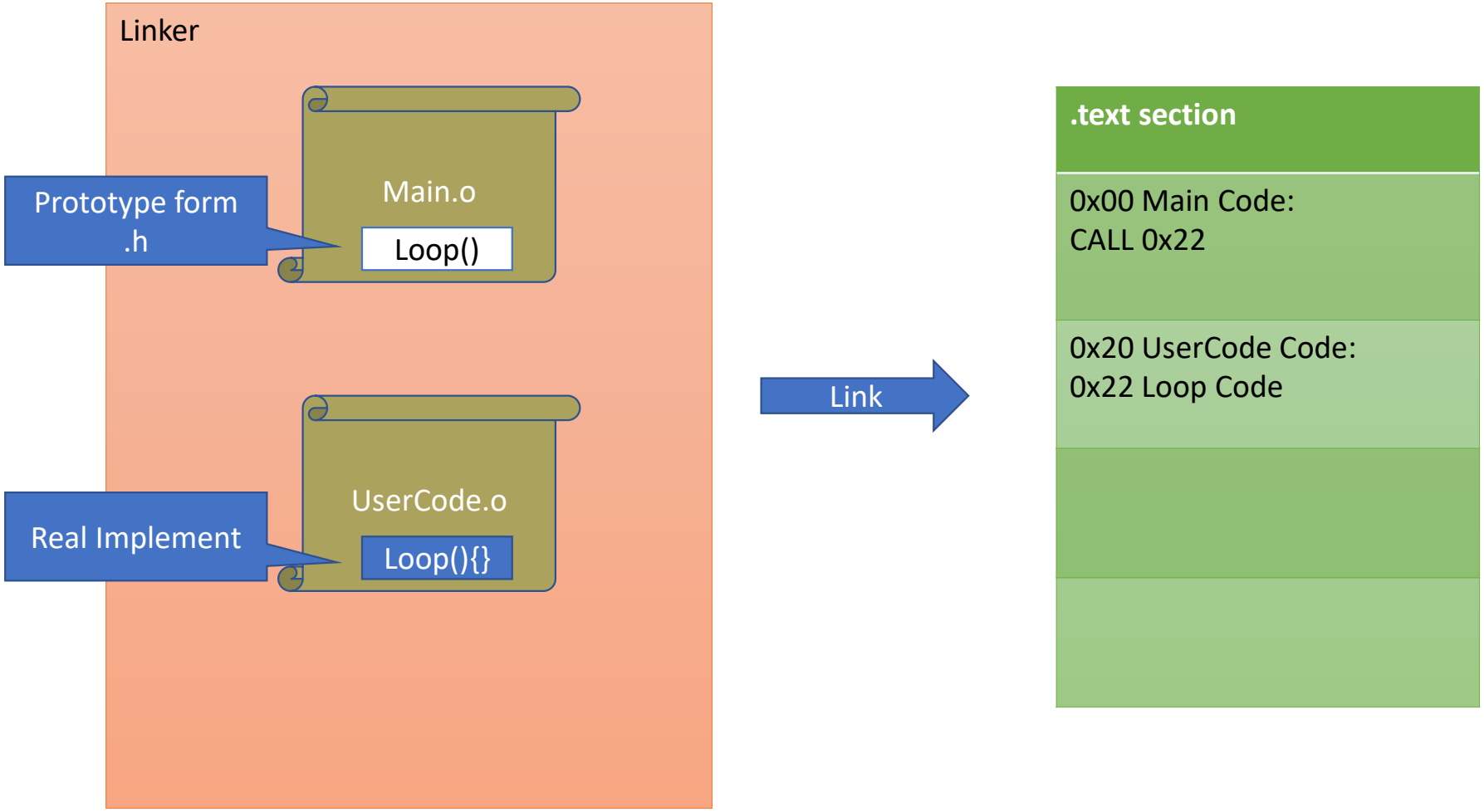
UserCode.c **Private implementation**
Documentation
Link section
Definition
Private Declaration Function Prototype
sub program

Code(in machine language)
+Data
Segmented with name tag

Linker


Compiler
main.c
int main
{
int A;
....
main.o
C0 22 FC 01 10 50 2B 4F 2D FF
FB 00 00 01 05 30 AD CA FB AA
23 58 94 35 12 00 45 00 00 00
00 12 35 8A 43 10 11 00 35 44
.....
Linker
main.o
C0 22 FC 01 10 50 2B 4F 2D FF
FB 00 00 01 05 30 AD CA FB AA
23 58 94 35 12 00 45 00 00 00
00 12 35 8A 43 10 11 00 35 44
.....
MCU Specific file
.elf
header
debug info
Unallocated memory
.data Section
.rodata Section
.text Section

Linker Map prototype to Real implement



main.c

```c
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "c2cpp.h"
/* USER CODE END Includes */
```

```c
/* USER CODE BEGIN 2 */
setup();
loop();
/* USER CODE END 2 */
```

c2cpp.h

```c
/*
 * c2cpp.h
 *
 *  Created on: May 11, 2022
 *      Author: AlphaP
 */

#ifndef INC_C2CPP_H_
#define INC_C2CPP_H_

void setup();
void loop();



#endif /* INC_C2CPP_H_ */
```

c2cpp.c

```c

#include "c2cpp.h"
void UserCodeSetup();
void UserCodeLoop();

void setup()
{
    //Call c++ domain
    UserCodeSetup();
}

void loop()
{
    //Call c++ domain
    UserCodeLoop();
}
```

Prototype declareation

UserCode.cpp

```c

#include "main.h"

extern "C" void UserCodeSetup();
extern "C" void UserCodeLoop();
extern UART_HandleTypeDef huart2;
void UserCodeSetup()
{

}

void UserCodeLoop()
{
    while(1)
    {
        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
        HAL_Delay(500);
    }
}
```

implement

Library – How it Work

- .h – Tell “What” this library have but should not include code implement
  - Because all implement will compile every time in every files that include this .h

- .C - Tell “How” this library Do
  - linker will link function call to code
  - code need compile just once.

Library

- .h ,.c don’t need to be same name (but keep it same name for understanding)
- linker need only .o / .a
  - we can share only .o and .h to other people that want to use our library and keep algorithm secret.
  - multiple .o can combine to .a to pack larger library
- in stm32cubeIDE(and many of IDE)
  - .c /.o automatically include to linker when create / import form IDE

Optimization -Basic

- Fast Math
- Variable Optimization
- function call optimization

Optimization -Fast Math

- multiply integer by 2,4,8,16,32,… use shift bits <<
- divide integer by 2,4,8,16,32,… use shift bits >>
  - complier may optimize this by itself

- Simplify Expression avoid division / Multiply
  - (a*b+a*c) ➔ a(b+c)
  - (pow(a,2) –(pow(b,2)) ➔ (a*a) - (b*b) ➔ (a+b)*(a-b)

$$Ax^5 + Bx^4 + Cx^3 + Dx^2 + Ex + F = \Big(\Big(\big((Ax + B) * x + C\big) * x + D\Big) * x + E\Big) * x + F$$

- in some case ++i faster than i++

Optimization -Fast Math

- Group up Constants
  - 5 *(3 *A) ➔ 15*A
- If it constanst , Make it constant ; #define PI 3.1415

- understand Instruction Set and timming

Table 7-1 FPU instruction set

Operation	Description	Assembler	Cycles
Absolute value	Of float	VABS.F32	1
Addition	Floating-point	VADD.F32	1
Compare	Float with register or zero	VCMP.F32	1
	Float with register or zero	VCMPE.F32	1
Convert	Between integer, fixed-point, half-precision and float	VCVT.F32	1
Divide	Floating-point	VDIV.F32	14
Load	Multiple doubles	VLDM.64	1+2*N, where
	Multiple floats	VLDM.32	1+N, where N
	Single double	VLDR.64	3
	Single float	VLDR.32	2
Move	top/bottom half of double to/from core register	VMOV	1
	immediate/float to float-register	VMOV	1
	Two floats/one double to/from two core registers or one float to/from one core register	VMOV	2
	floating-point control/status to core register	VMRS	1
	Core register to floating-point control/status	VMSR	1
Multiply	Float	VMUL.F32	1
	Then accumulate float	VMLA.F32	3
	Then subtract float	VMLS.F32	3
	Then accumulate then negate float	VNMLA.F32	3
	Then subtract then negate float	VNMLS.F32	3

Optimization - variable

- if we have space, use uint32_t / int32_t
- if we have array of structure, keep structure size in multiply by 2
- minimize local non static variable ()
- declare structure variable in align with memory
- Masking faster than bitfield

Optimization - variable

- avoid dereference (*a) too much , if it need for calculate process , copy to temp variable

- LUT is Good for some problem

Optimization function call optimization

- minimize parameter
  - if too many parameter , use structure to pass value , and pass it by reference
- Void function if no return / return by reference
- use inline in short function
- avoid too much Interrupt
- Use Library for grouping function together