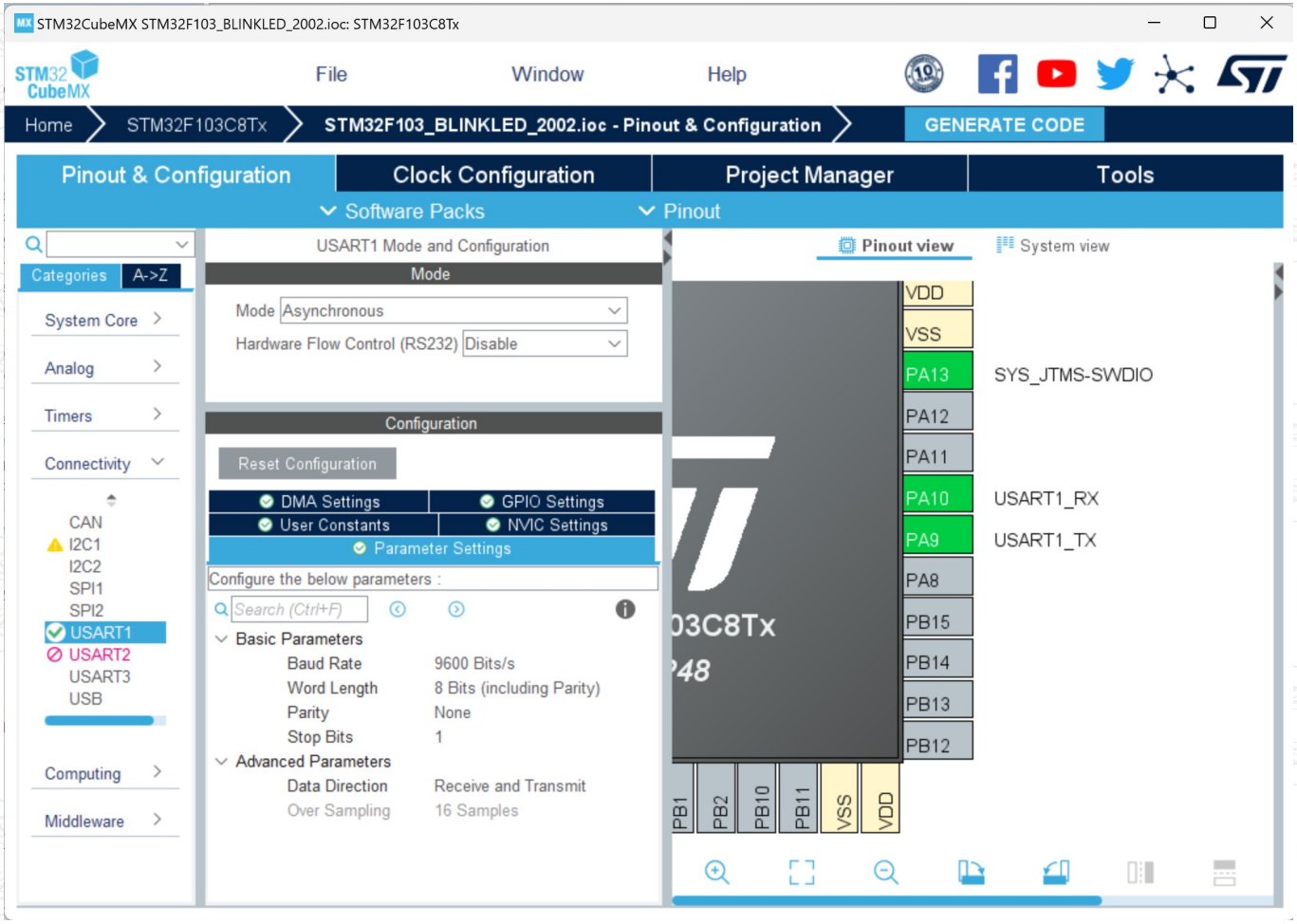
2.4. Debug

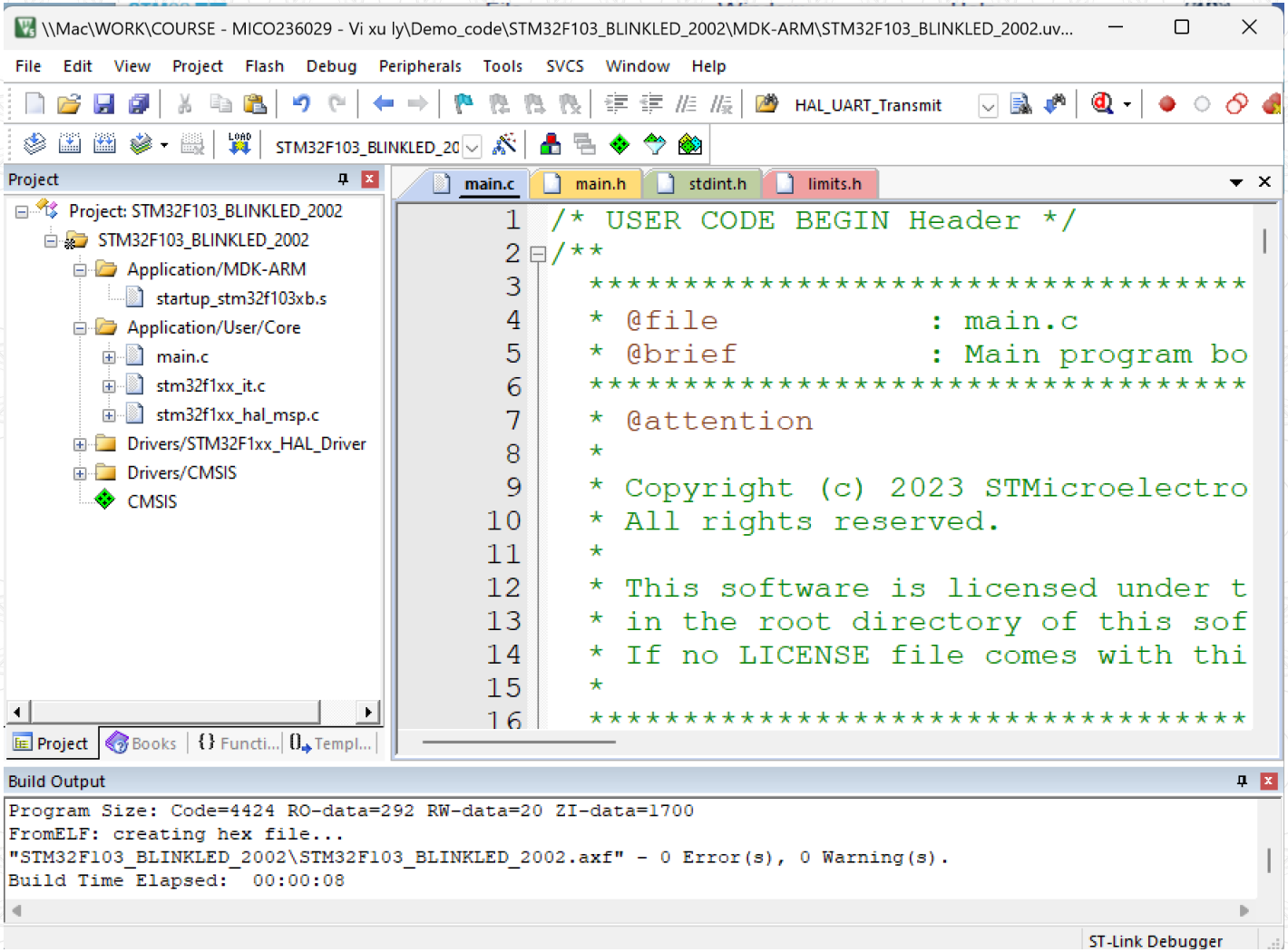
Bước 1: Tải và cài đặt phần mềm STM32 CubeMX

https://www.st.com/en/development-tools/stm32cubemx.html



2.4. Debug

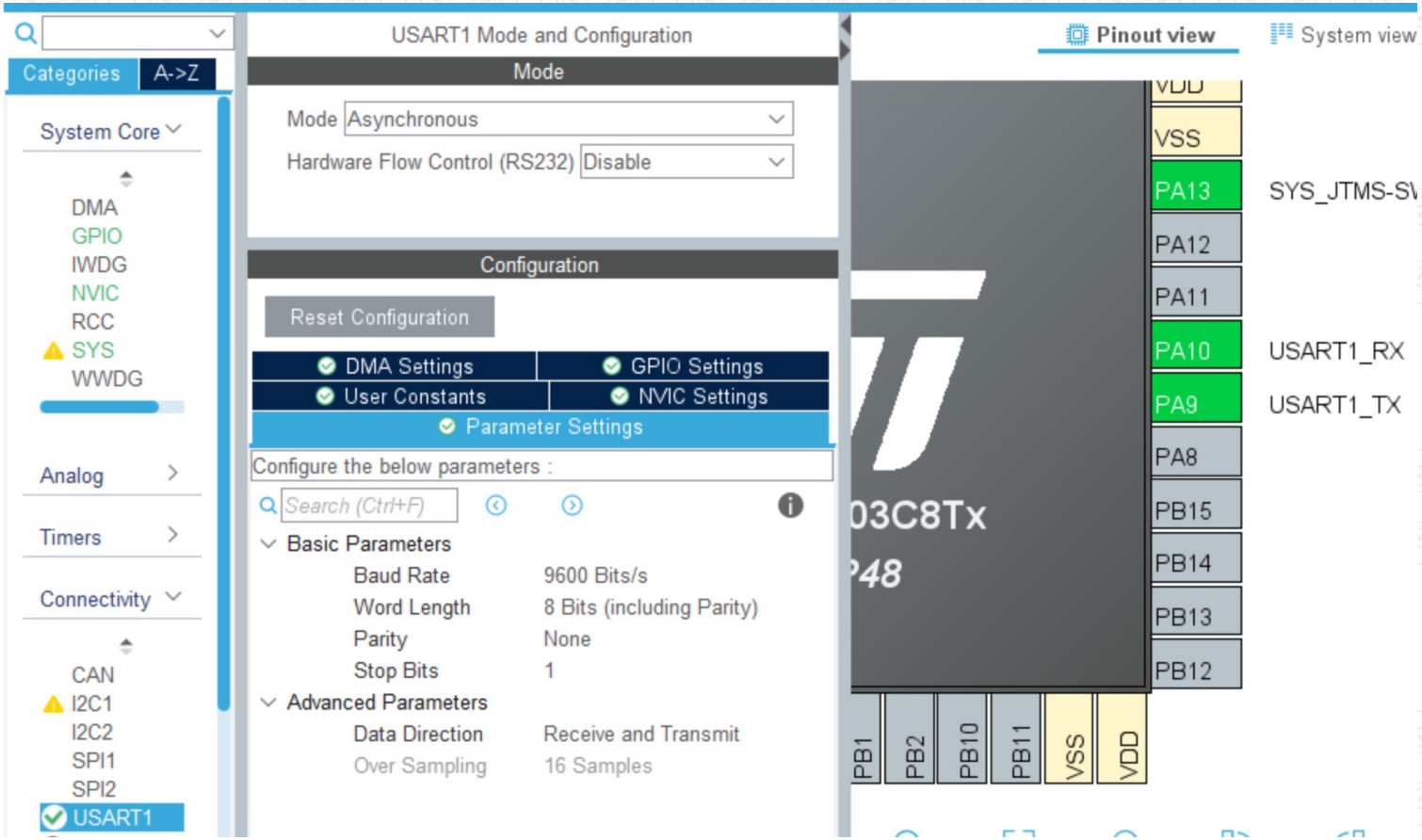
Bước 2: Tải và cài đặt phần mềm KeilC



2.4. Debug

Bước 3: Khởi tạo dự án trên STM32 CubeMX

- STM32F103C8
- USART1



2.4. Debug

Bước 4: Lập trình HAL_UART_Transmit

HAL_UART_Transmit

Function name

HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Sends an amount of data in blocking mode.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

```
uint8_t s2[]={72,69,76,76,79,32,87,79,82,76,68};
while (1)
{
  HAL_UART_Transmit(&huart2,s2,11,100);
  HAL_Delay(500);
}
```

2.4. Debug

Bước 5: Retargeting **printf** trong KeilC

```
struct __FILE
{
  int handle;

};

FILE __stdout;
int fputc(int ch, FILE *f)
{
  char tempch = ch;

  HAL_UART_Transmit(&huart1, (uint8_t *)&tempch,1,100);
  return ch;
}
```

```
while (1)
{
  printf("Hello world\n");
  HAL_Delay(500);
}
```

2.4. Debug

Bước 5: Sử dụng **printf** để debug

%[**flags**][**width**][**.precision**][**length**]**specifier**

specifier	argument
d or i	Signed INT
u	Unsigned INT
o	Unsigned Octal
x or X	Unsigned Hexadecimal (lower/upper case)
e or E	Floating Point
g or G	Shortest Representation
a or A	Hexadecimal Floating Point (lower/upper case)
c	Character
s	String of characters
p	Pointer address

flag	description
-	Left-justify (default: right)
+	Force print + sign with positives
(space)	Add space, if no sign before value
#	Precede o, x, or X with 0
0	Left pad number with zeros

.precision	description
.number	specifies number of digits to write
.*	not specified in cstring, additional INT value given

width	description
(number)	Minimum number of characters to print
*	Not specified in cstring, additional INT value given

2.4. Debug

Bước 5: Sử dụng **printf** để debug

%[flags][width][.precision][length]specifier

```
printf ("Integers: %i %u \n", -3456, 3456);
printf ("Characters: %c %c \n", 'z', 80);
printf ("Decimals: %d %ld\n", 1997, 32000L);
printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
printf ("floats: %4.2f %+.0e %E \n", 3.14159, 3.14159, 3.14159);
printf ("Preceding with empty spaces: %10d \n", 1997);
printf ("Preceding with zeros: %010d \n", 1997);
printf ("Width: %*d \n", 15, 140);
printf ("%s \n", " A string ");
```

```
int ch;
for( ch = 75 ; ch <= 100; ch++ ) {
   printf("ASCII value = %d, Character = %c\n", ch , ch );
}
```

2.4. Debug

Bài tập: Tạo dự án và lập trình debug để in ra Terminal đồng hồ đếm **giờ : phút : giây**

Ví dụ:

08:59:56
08:59:57
08:59:58
08:59:59
09:00:00
09:00:01
...

2.8. Ôn tập về ngôn ngữ C: Basic types

There are only a few basic data types in C:

- **char** = a single byte, capable of holding one character
- **int** = an integer
- **float** = single-precision floating point
- **double** = double-precision floating point

But in C++, they introduced int8_t , int16_t, uint8_t, etc…

```
    /* exact-width signed integer types */
typedef   signed          char int8_t;
typedef   signed short     int int16_t;
typedef   signed           int int32_t;
typedef   signed       __INT64 int64_t;

    /* exact-width unsigned integer types */
typedef unsigned          char uint8_t;
typedef unsigned short     int uint16_t;
typedef unsigned           int uint32_t;
typedef unsigned       __INT64 uint64_t;
```

stdint.h

2.8. Ôn tập về ngôn ngữ C : Basic types

```
    /* minimum values of exact-width signed integer types */
#define INT8_MIN                         -128
#define INT16_MIN                      -32768
#define INT32_MIN                (~0x7fffffff)   /* -2147483648 is unsigned */
#define INT64_MIN   __INT64_C(~0x7fffffffffffffff)
                         /* -9223372036854775808 is unsigned */
```

```
    /* maximum values of exact-width signed integer types */
#define INT8_MAX                          127
#define INT16_MAX                       32767
#define INT32_MAX                  2147483647
#define INT64_MAX   __INT64_C(9223372036854775807)
```

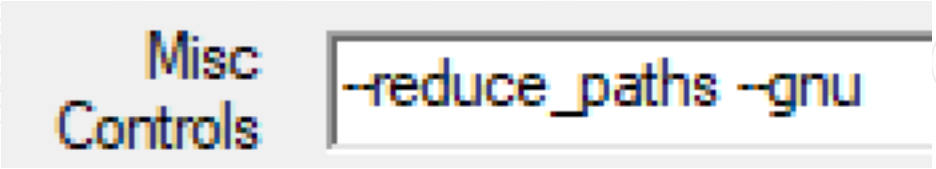
```
    /* maximum values of exact-width unsigned integer types */
#define UINT8_MAX                         255
#define UINT16_MAX                      65535
#define UINT32_MAX                 4294967295u
#define UINT64_MAX  __UINT64_C(18446744073709551615)
```

Bài tập: Viết chương trình debug tìm và in ra số giờ hạn của **int8_t, int16_t, int32_t, int64_t, uint8_t, uint16_t, unint32_t và uint64_t**

2.8. Ôn tập về ngôn ngữ C : Constant and enum

Khi viết **hằng số**:

- Số nguyên bình thường: 1234 kiểu **int**
- Muốn đổi kiểu lớn hơn thì thêm hậu tố: 1234**L** hoặc 1234**l** (kiểu long); hoặc 123**ul**, 123**UL** (unsigned long)
- Thêm hậu tiền tố để chỉ hệ đếm: 0x2f, 0x2F hoặc 0X2F, 0X2f để chỉ kiểu số HEX
- Muốn gọi kí tự trong bảng mã ASCII thì dùng tiền tố \ trước số thứ tự của ký tự cần gọi. Ví dụ: \65 (“A”)
- Muốn biểu thị dạng nhị phân **0B**11001010 hoặc 0**b**11001101 thì cần thêm chỉ định **--gnu** vào **Misc Controls** trong **Option for target >> C/C++**



2.8. Ôn tập về ngôn ngữ C: Constant and enum

Tập hợp hằng số tự định nghĩa **enum**

enum boolean { NO, YES };

Khai báo tập hợp hằng số tên là boolean với giá trị 0 được đặt tên là NO; giá trị 1 (tiếp theo) được đặt tên là YES.

```
enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};
int main() {
    enum week day;
    day = Wed;                    // day = 2
    printf("%d",day);
    return 0;
}
```

Bài tập:
Tìm định nghĩa enum **GPIO_PinState**

```
enum day {sunday = 1, monday, tuesday = 5,
          wednesday, thursday = 10, friday, saturday};
printf("%d %d %d %d %d %d %d", sunday, monday, tuesday,
            wednesday, thursday, friday, saturday);
```

2.8. Ôn tập về ngôn ngữ C: Constant and enum

```
// implementation 1
int Mode;  // 0 means error
void function1(void){
  Mode = 1; // no error
}
void function2(void){
  if(Mode == 0){ // error?
  UART_OutString("error");
  }
}
```

```
// implementation 2
#define NOERROR 1
#define ERROR 0
int Mode;
void function1(void){
  Mode = NOERROR;
}
void function2(void){
  if(Mode == ERROR){
  UART_OutString("error");
  }
}
```

2.8. Ôn tập về ngôn ngữ C: Constant and enum

```
// implementation 3
const int NOERROR = 1;
const int ERROR = 0;
int Mode;
void function1(void){
  Mode = NOERROR;
}
void function2(void){
  if(Mode == ERROR){
 UART_OutString(“error”);
  }
}
```

```
// implementation 4
enum Mode_state{ ERROR,
  NOERROR};
enum Mode_state Mode;
void function1(void){
  Mode = NOERROR;
}
void function2(void){
  if(Mode == ERROR){
  UART_OutString(“error”);
  }
}
```

2.8. Ôn tập về ngôn ngữ C : Conditional and loop

```
if (expression)
    statement1
else
    statement2
```

```
while (expression)
    statement
```

```
for (expr1; expr2; expr3)
    statement
```

```
if (expression)
    statement
else if (expression)
    statement
else if (expression)
    statement
else if (expression)
    statement
else
    statement
```

```
do
    statement
while (expression);
```

Bài tập:

Vẽ lưu đồ cho các lệnh lặp vòng

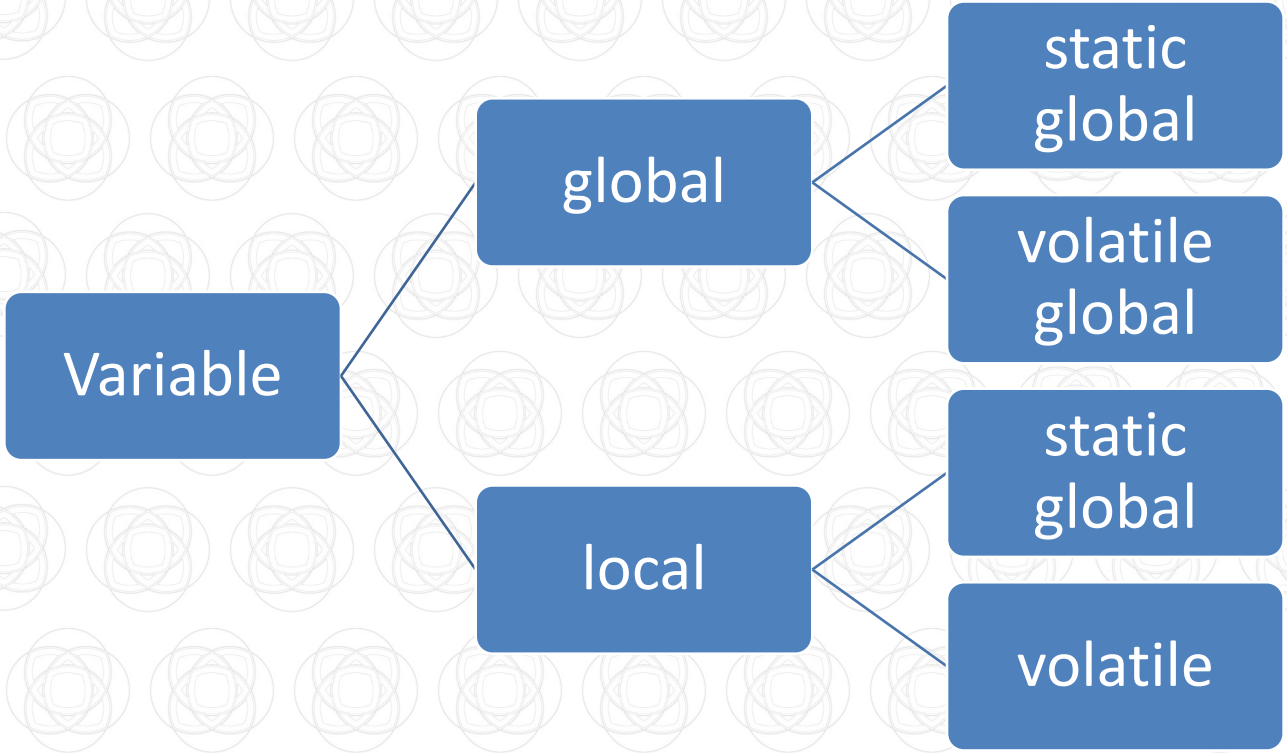
2.8. Ôn tập về ngôn ngữ C: Variable scope

```
int c= 30; /* global area */
main ( ) {
        int a = 10; //local area
        printf ("a=%d, c=%d", a,c);
        fun ( );
}
fun ( ){
        printf ("c=%d",c);
}
```

Biến toàn cục khai báo **trước** hàm main() và có tác dụng trong tất cả chương trình chính và các hàm con

Biến cục bộ khai báo trong hàm. Được khởi tạo khi có gọi hàm và bị xoá khi kết thúc gọi hàm.

2.8. Ôn tập về ngôn ngữ C: Variable scope



2.8. Ôn tập về ngôn ngữ C: Variable scope

```
#include <stdio.h>
void fun() {
  static int x; //default is 0
  printf("%d ", x);
  x = x + 1; }
int main(){
  fun();  fun();
  return 0;
}
```

static thêm vào trước khi khai báo biến để chỉ định biến **không** bị xoá khi kết thúc hàm.

```
void function1(void){
  static short TheCount;
  TheCount = TheCount+1;
}
void function2(void){
  static short TheCount;
  TheCount = TheCount+1;
}
```

static báo trình dịch sử dụng biến riêng của file khai báo nó, không có giá trị ở file khác.

2.8. Ôn tập về ngôn ngữ C: Variable scope

```
volatile unsigned long Time;
void SysTick_Handler(void) {
      Time = Time++;
}
void main(void) {
     SysTick_Init();
     Time = 0;
     while(Time<100) {};
```

Thêm **volatitle** giúp biến được nạp lại giá trị từ ram mỗi khi chương trình sử dụng tới biến đó.

=> Sử dụng cho ô nhớ liên kết ngoại vi (mapped I/O) hoặc ngắt

```
unsigned char data[100];
#define GPIO_PORTA_DATA_R       (*((volatile unsigned long *)0x400043FC))
void Collect(void){ short i;
  for(i=0;i<100;i++){ /* collect 100 measurements */
    data[i] = GPIO_PORTA_DATA_R;  /* collect ith measurement */
  }
}
```

2.8. Ôn tập về ngôn ngữ C: Variable scope

```
#define  GPIOA          ((GPIO_TypeDef *)GPIOA_BASE)
typedef struct
{
  __IO uint32_t CRL;
  __IO uint32_t CRH;
  __IO uint32_t IDR;
  __IO uint32_t ODR;
  __IO uint32_t BSRR;
  __IO uint32_t BRR;
  __IO uint32_t LCKR;
} GPIO_TypeDef;
#define    __IO    volatile          /*!< Defines 'read / write' permissions */
```

Ví dụ thư viện HAL của STM32 khi khai báo thanh ghi (biến) GPIO có sử dụng volatile

Khi khai báo **GPIOA**, thư viện cần thêm **volatitle** để trong chương trình chính, mỗi lần truy cập thanh ghi **GPIOA** thì các trạng thái mới (do phần cứng thay đổi GPIOA) được cập nhật.

2.8. Ôn tập về ngôn ngữ C: Variable scope

Thêm **extern** để đánh dấu biến được sử dụng ở tất cả các file trong dự án

```
extern short ExtGlobal;   /* an external global variable*/
void main(void){
    ExtGlobal=1000;
}
```

```
unsigned char x;   /* a regular global variable*/
void sub(void){
    x=1;
    {   unsigned char x;   /* a local variable*/
        x=2;
        {   unsigned char x;  /* a local variable*/
            x=3;
            PORTA=x;}
        PORTA=x;}
    PORTA=x;}
}
```