B1
Miscellaneous

C Program Structure

```
#include "STM32   xx.h"      /* I/O port/register names/addresses for the STM32      microcontrollers */

/* Global variables – accessible by all functions */
int count, bob;              //global (static) variables – placed in RAM

/* Function definitions*/
int function1(char x) {      //parameter x passed to the function, function returns an integer value
  int i,j;                   //local (automatic) variables – allocated to stack or registers
  -- instructions to implement the function
}

/* Main program */
void main(void) {
   unsigned char sw1;        //local (automatic) variable (stack or registers)
   int k;                    //local (automatic) variable (stack or registers)
  /* Initialization section */
   -- instructions to initialize  variables, I/O ports, devices, function registers
  /* Endless loop */
   while (1) {               //Can also use:  for(;;) {
    -- instructions to be repeated
   } /* repeat forever */
}
```

Declare local variables

Initialize variables/devices

Body of the program

Review of basic concepts

```c
#include<stdio.h>
int fun()
{
  int count = 0;
  count++;
  return count;
}

int main()
{
  printf("%d ", fun());
  printf("%d ", fun());
  return 0;
}
```

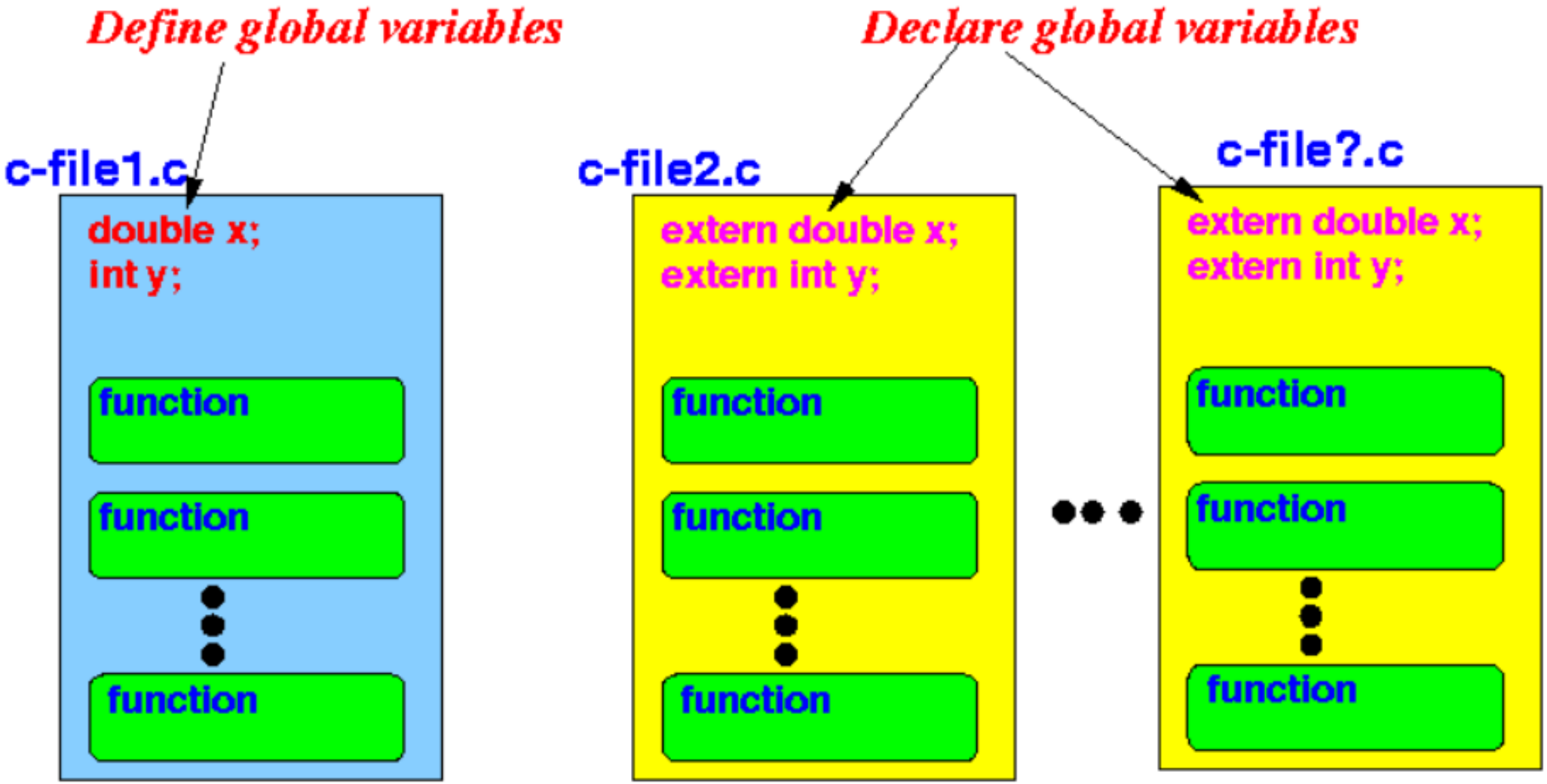
Output: **1 1**

```c
#include<stdio.h>
int fun()
{
  static int count = 0;
  count++;
  return count;
}

int main()
{
  printf("%d ", fun());
  printf("%d ", fun());
  return 0;
}
```

Output: **1 2**

Review of basic concepts



Review of basic concepts

- **Header file in C**: A header file is a file with extension .h which contains **C function declarations** and macro **definitions** to be shared between several source files. There are two types of header files: the files that the programmer writes and the files that comes with your tool chain (compiler, linker, etc).
- Common error: call header files as libraries!!!!
- **Library file**: file that **contains the function code** for the declared functions in the header file

Data types in C
(see also stdint.h)

Type	Size in bits	Natural alignment in bytes	Range of values
Char/int8_t	8	1 (byte-aligned)	0 to 255 (unsigned) by default
signed char/int8_t	8	1 (byte-aligned)	–128 to 127
unsigned char/uint8_t	8	1 (byte-aligned)	0 to 255
(signed) short	16	2 (halfword-aligned)	–32,768 to 32,767
unsigned short	16	2 (halfword-aligned)	0 to 65,535
(signed) int	32	4 (word-aligned)	–2,147,483,648 to 2,147,483,647
unsigned int	32	4 (word-aligned)	0 to 4,294,967,295
(signed) long	32	4 (word-aligned)	–2,147,483,648 to 2,147,483,647
unsigned long	32	4 (word-aligned)	0 to 4,294,967,295
(signed) long long	64	8 (doubleword-aligned)	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	64	8 (doubleword-aligned)	0 to 18,446,744,073,709,551,615
float	32	4 (word-aligned)	1.175494351e-38 to 3.40282347e+38 (normalized values)
double	64	8 (doubleword-aligned)	2.22507385850720138e-308 to 1.79769313486231571e+308 (normalized values)
long double	64	8 (doubleword-aligned)	2.22507385850720138e-308 to 1.79769313486231571e+308 (normalized values)
wchar_t	16	2 (halfword-aligned)	0 to 65,535 by default.
	32	4 (word-aligned)	0 to 4,294,967,295 when compiled with --wchar32.
All pointers	32	4 (word-aligned)	Not applicable.
bool (C++ only)	8	1 (byte-aligned)	false or true
_Bool (C only[a])	8	1 (byte-aligned)	false or true

C operators

- Logical Operators

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c==5) || (d>5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `!(c==5)` equals to 0.

C operators

- Bitwise Operators

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

C operators

&	AND
|	OR
^	XOR
~	One's Compliment 0 → 1 1 → 0
<<	Left shift
>>	Right Shift

- Bitwise Operators. Examples
  - Let us suppose two 8 bits integers: **A=12** and **B=25** (decimal)

    A = 00001100 in binary

    B = 00011001 in binary

Perform the following calculations:

- A&B = ???
- A|B = ???
- A^B = ???
- ~A = ???
- A<<3 = ???
- B<<5 = ???
- A>>3 = ???
- B>>5 = ???

C operators

&	AND
|	OR
^	XOR
~	One's Compliment 0 → 1 1 → 0
<<	Left shift
>>	Right Shift

- Bitwise Operators. Examples
  - Let us suppose two 8 bits integers: **A=12** and **B=25** (decimal)

    A = 00001100 in binary

    B = 00011001 in binary

Perform the following calculations:

- A&B = 00001000 **LOGIC AND: COMMON '1'**
- A|B = 00011101 **LOGIR OR: ALL '1' in both**
- A^B = 00010101 **LOGIC XOR: Every different bit ->'1'**
- ~A = 11110011 **COMPLEMENTO 1: Invert**
- A<<3 = 01100000 **LSHIFT 3: Desplazamiento izda 3. == *2^3 == *8**
- B<<5 = 00100000 **LSHIFT 5: Desplazamiento izda 5. No se rellena nada y se pierden '1'**
- A>>3 = 00000001 **RSHIFT 3: Desplazamiento dcha 3 binario (sin relleno)**
- B>>5 = 00000000 **LSHIFT 3: Desplazamiento izda 5 binario**