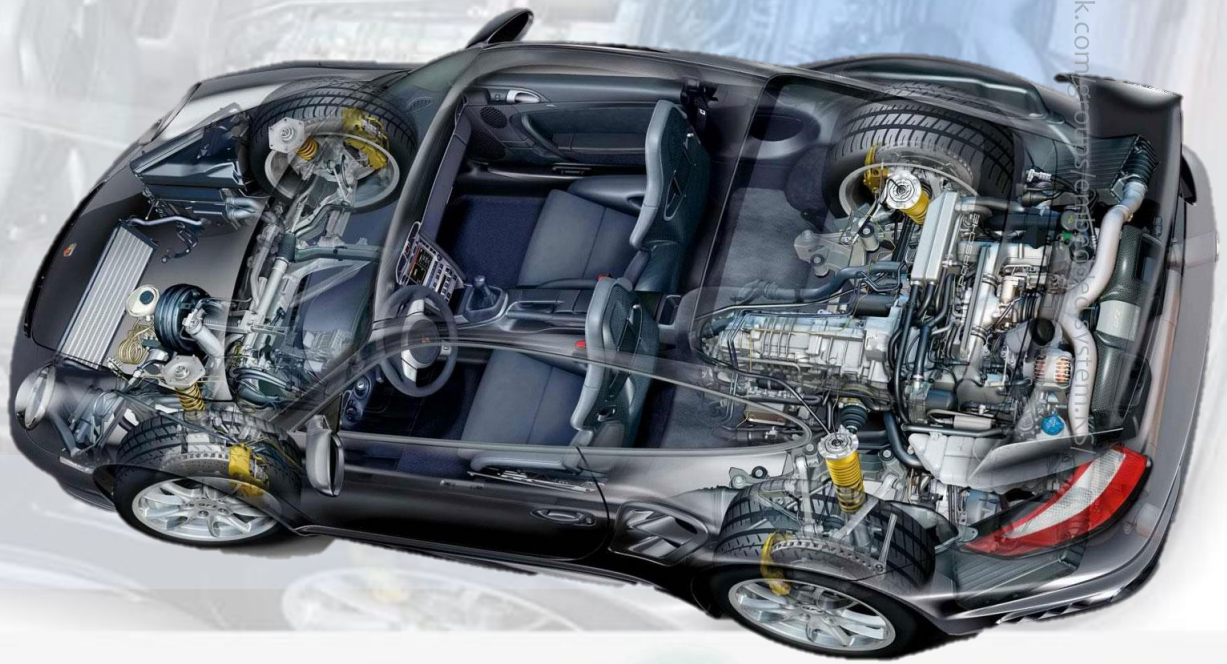
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Unit2 Lesson 7
C Programming)

- Macros
- #pragma
- Constant



ENG.KEROLES SHENOUDA

C – Preprocessor directives

- Before **a C program is compiled** in a compiler, source code is processed by a program called **preprocessor**. This process is called **preprocessing**.
- Commands used in preprocessor are called **preprocessor directives** and they begin with "#" symbol.

A program in C language involves into different processes

Program process flow	File name in each steps	Description
Source code	sample.c	
↓ Preprocessor		→ Preprocessor replaces #define (macro), #include (files), conditional compilation codes like #ifdef, #ifndef by their Respective values & source codes in source file
Expanded source code	sample.i	
↓ Compiler		→ Compiler compiles expanded source code to assembly source code
Assembly source code	sample.s	
↓ Assembler		→ It is a program that converts assembly source code to object code.
Object code	sample.o	
↓ Linker		→This is a program that converts object code to executable code and also combines all object codes together.
Executable code	sample.exe	
↓ Loader		→ Executable code is loaded in CPU and executed by loader program.
Execution		

DIFFERENCE BETWEEN COMPILERS VS INTERPRETERS IN C LANGUAGE?

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Compilers	Interpreters
Compiler reads the entire source code of the program and converts it into binary code. This process is called compilation. Binary code is also referred as machine code, executable, and object code.	Interpreter reads the program source code one line at a time and executing that line. This process is called interpretation.
Program speed is fast.	Program speed is slow.
One time execution. Example: C, C++	Interpretation occurs a every line of the program. Example: BASIC

```
MINGW64:/d/courses/new_diploma/Diploma online/C/macro_labs

kkhalil@egc-kkhalil-lt MINGW64 /d/courses/new_diploma/Diploma online/C/macro_labs
$ ./script.sh
Enter the first number : Enter the second number : sum of two value is 2
Execution time was 0 seconds.

kkhalil@egc-kkhalil-lt MINGW64 /d/courses/new_diploma/Diploma online/C/macro_labs
$
```

Windows PowerShell

```
Windows PowerShell
PS C:\MinGW\bin> .\main.exe
Enter the first number :
Enter the second number :
sum of two value is 3
fun() took 0.001000 seconds to execute
PS C:\MinGW\bin>
```

lab_macros.txt | script.sh | main.c — D\...\macro_labs | main.c

```
#!/bin/bash
#//@ www.learn-in-depth.com

start=`date +%s`

echo -n "Enter the first number : "
#read num1
num1=1
echo -n "Enter the second number : "
 num2=1
#read num2

sum=`expr $num1 + $num2`
echo "sum of two value is $sum"

end=`date +%s`
echo Execution time was `expr $end - $start` seconds.
```

```
//@ www.learn-in-depth.com
#include "stdio.h"
#include <time.h>

// clock() function which is available time.h.
// CLOCKS_PER_SEC (the number of clock ticks per second)
void main ()
{
    clock_t t;
    t = clock();
    int num1 , num2 ,sum ;
    printf ("Enter the first number : \n");
//  scanf ("%d",&num1);
    num1 = 1 ;
    printf ("Enter the second number : \n");
//  scanf ("%d",&num2);
    num2 = 2 ;
    sum = num1 + num2 ;
    printf("sum of two value is %d\n",sum );
     t = clock() - t;
      double time_taken = ((double)t)/CLOCKS_PER_SEC; // in second
     printf("fun() took %f seconds to execute \n", time_taken);
}
```

ook.com/groups/embedded.system.KS/

Pre-processor directives in C

Pre-processor directives in C

- In C programming pre-processor directives are used to affect **compile-time** settings
- Pre-processor directives begin with **"#"** symbol
- Pre-processor directives are **resolved** or ***taken*** **cared** during the

pre-processing stage of compilation

list of preprocessor directives that C programming language offers.

Preprocessor	Syntax/Description
Macro	**Syntax: #define** This macro defines constant value and can be any of the basic data types.
Header file inclusion	**Syntax: #include <file_name>** The source code of the file “file_name” is included in the main program at the specified place.
Conditional compilation	**Syntax: #ifdef, #endif, #if, #else, #ifndef** Set of commands are included or excluded in source program before compilation with respect to the condition.
Other directives	**Syntax: #undef, #pragma, #error** #undef is used to undefine a defined macro variable. #Pragma is used to call a function before and after main function in a C program.

Macro

Macros are written in C using **# define pre-processor** directives

Macros are **text replacement** in code

No semicolon is needed to end a macro definition

Syntax

`#define` *identifier replacement-list* (optional)	(1)	
`#define` *identifier*`(` *parameters* `)` *replacement-list*	(2)	
`#define` *identifier*`(` *parameters*`, ...` `)` *replacement-list*	(3)	(since C99)
`#define` *identifier*`(` `...` `)` *replacement-list*	(4)	(since C99)
`#undef` *identifier*	(5)	

Macro

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Project Explorer

- C_session_1_printf_tricks
- C_Session1_typeCasting
- first_c_code
- lkl
- second_session_c
- session2
  - Binaries
  - Includes
  - Debug
  - main.c
  - makefile

main.c

```c
//Prepared by Eng.Keroles

#include <stdio.h>

#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '?'

int main(int argt ,char**argv){
    printf("value of height    : %d \n", height );
    printf("value of number : %f \n", number );
    printf("value of letter : %c \n", letter );
    printf("value of letter_sequence : %s \n", letter_sequence);
    printf("value of backslash_char  : %c \n", backslash_char);

}
```

Macros
Is
Text Replacement

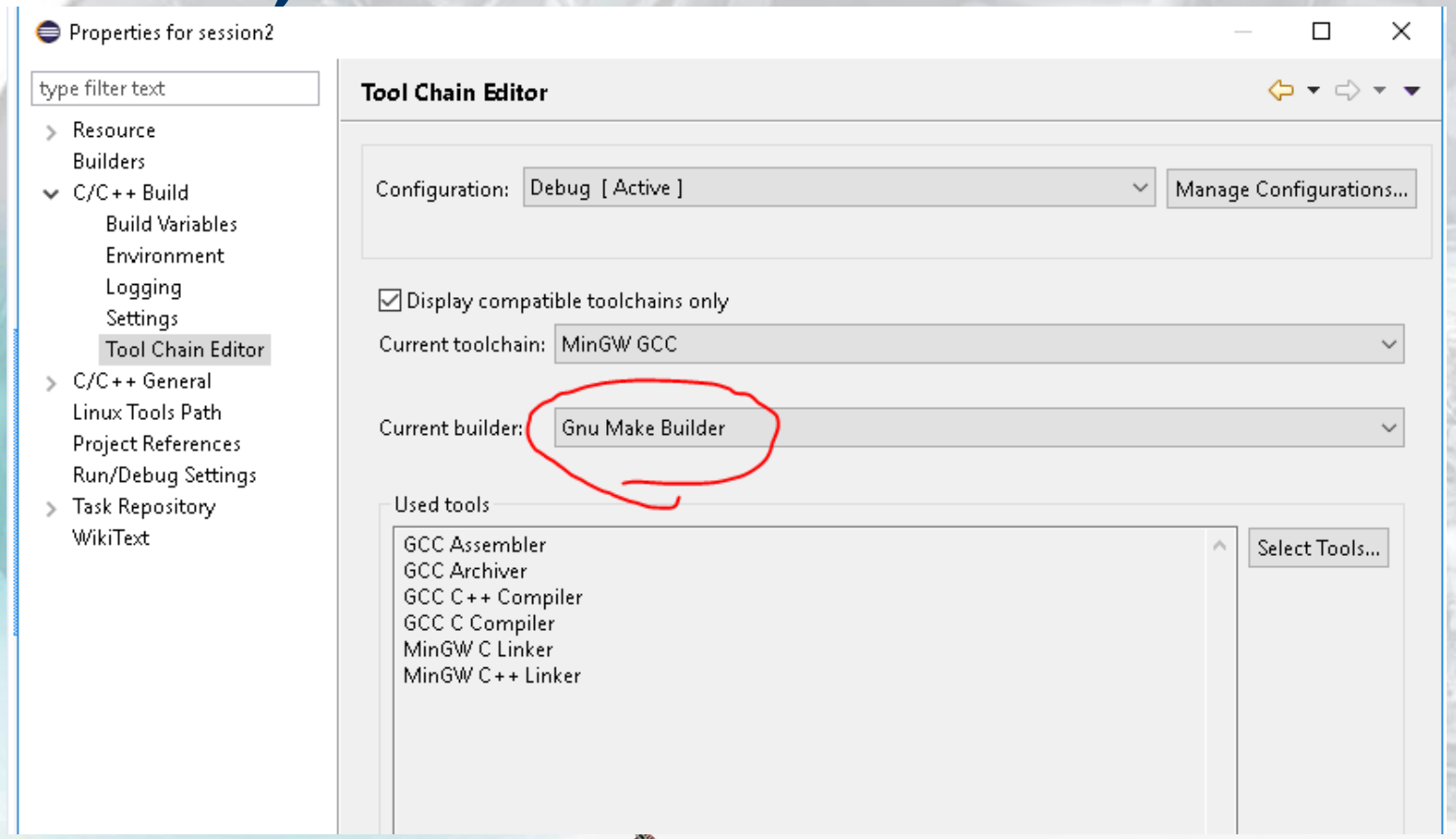
Problems | Tasks | Console | Properties | AVR De

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\cours

```
value of height    : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char  : ?
```

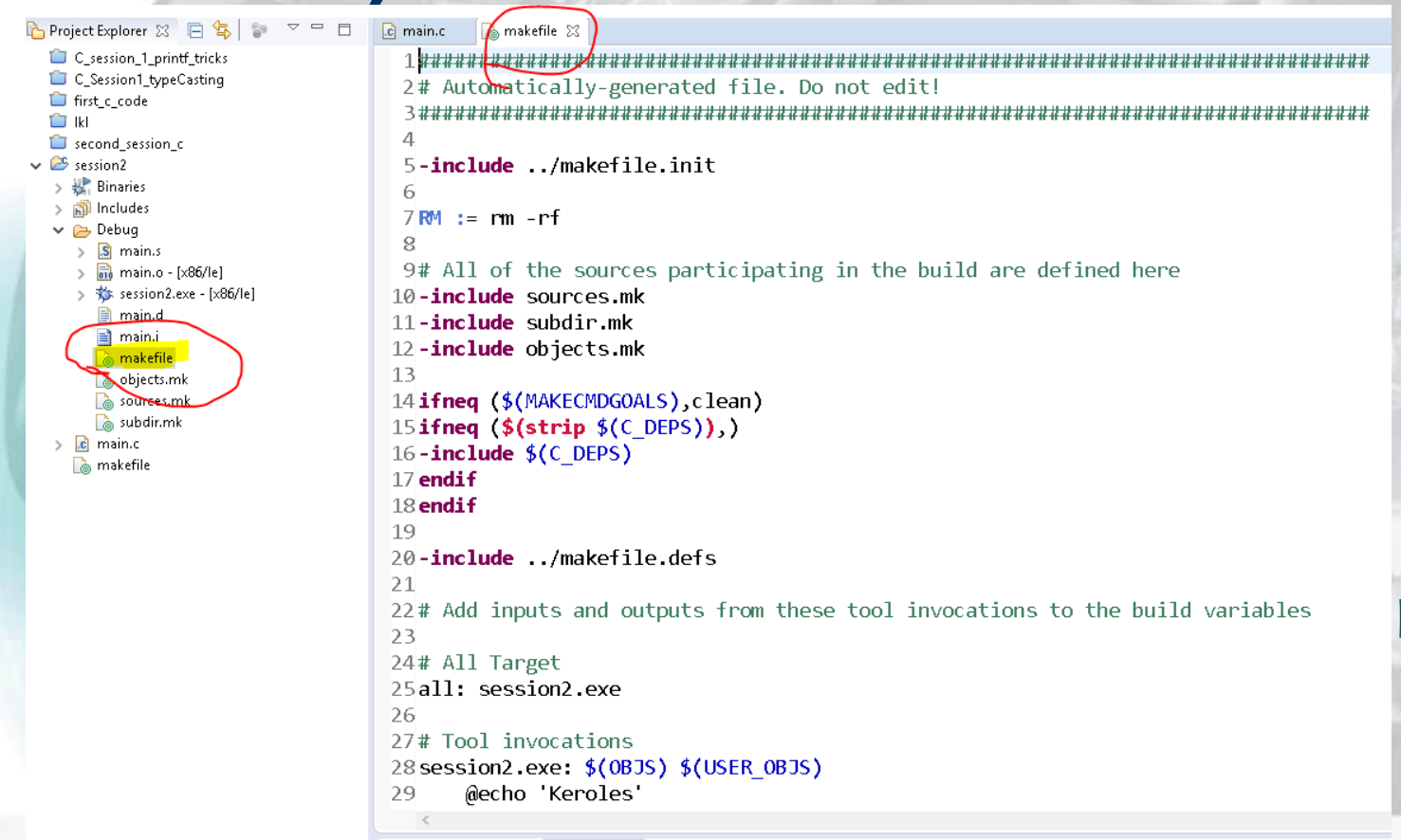
https://www.facebook.com/groups/embedded.system.KS/

Macro
How To see the .i file (precompiled file)



First we will
Modify the Project Settings
To Compile not internally
But from external Makefile

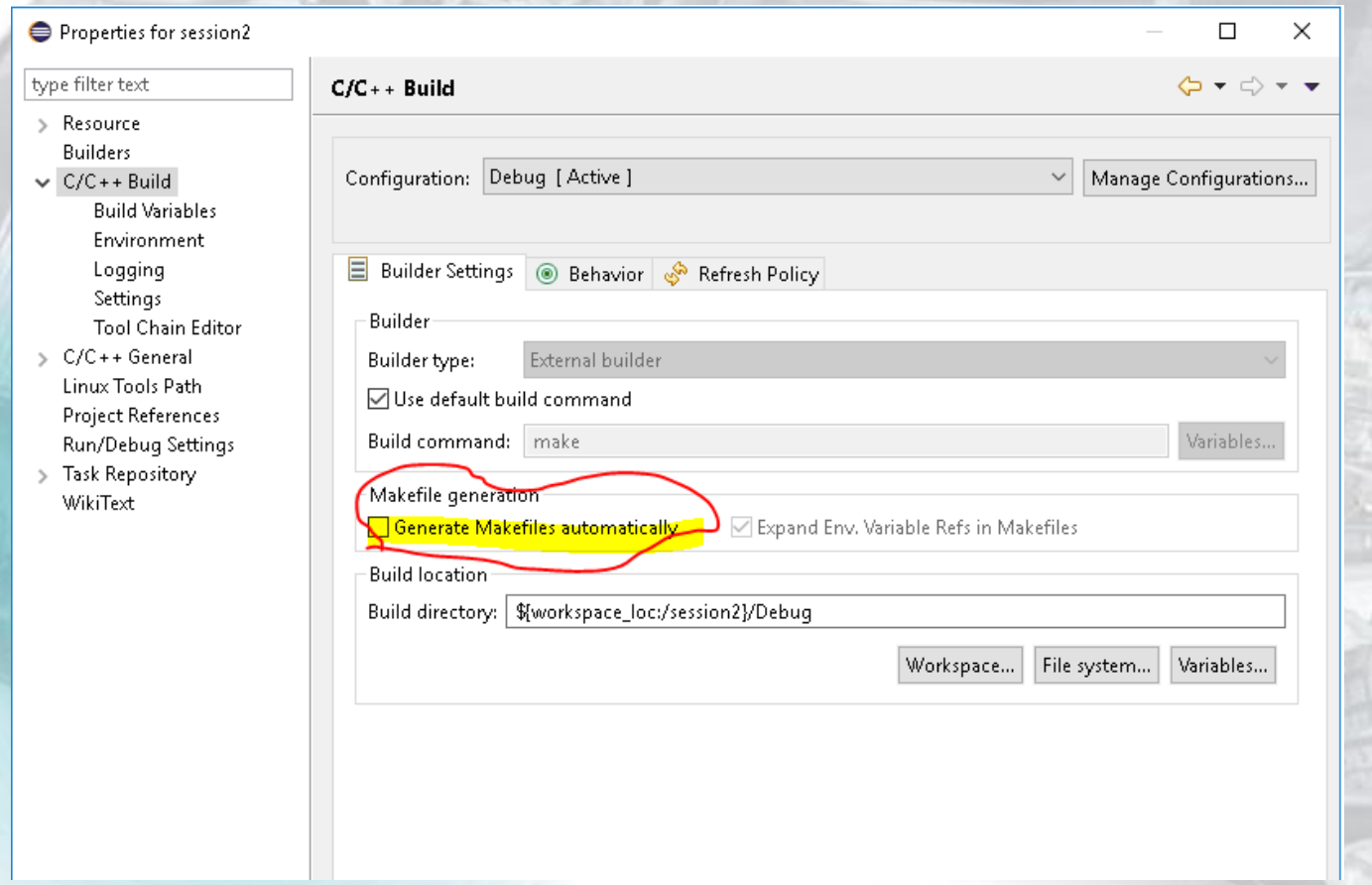
Macro
How To see the .i file (precompiled file)



Second
Run the Build and Then you will Find a Makefile will be created Automatically

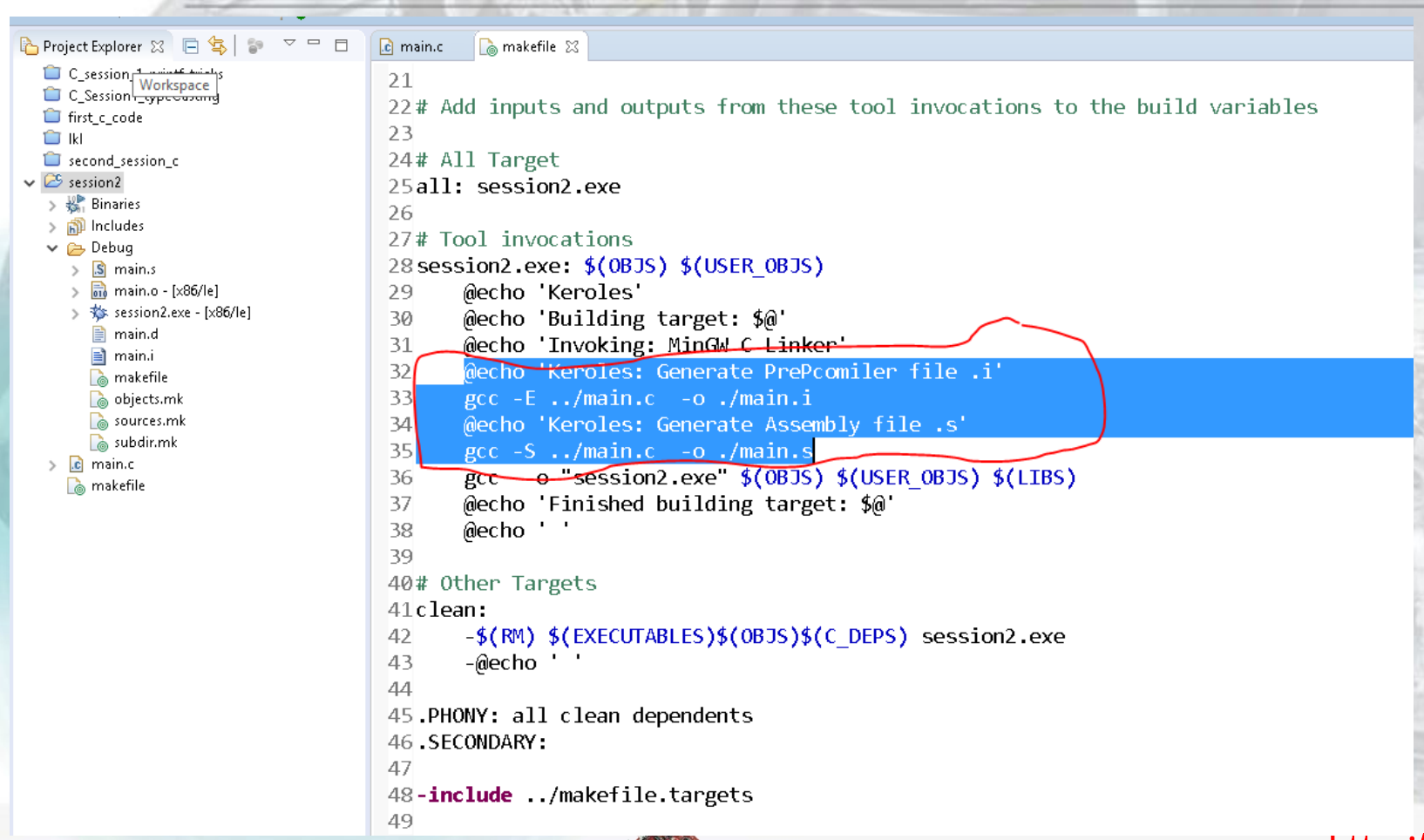
Kindly Note You will get full Knowledge on Makefile on the Next Sessiones

Macro
How To see the .i file (precompiled



Third: This Makefile generated Automatically at Every time you build So if we want to Modify on it without any change Do the Following Unmark it

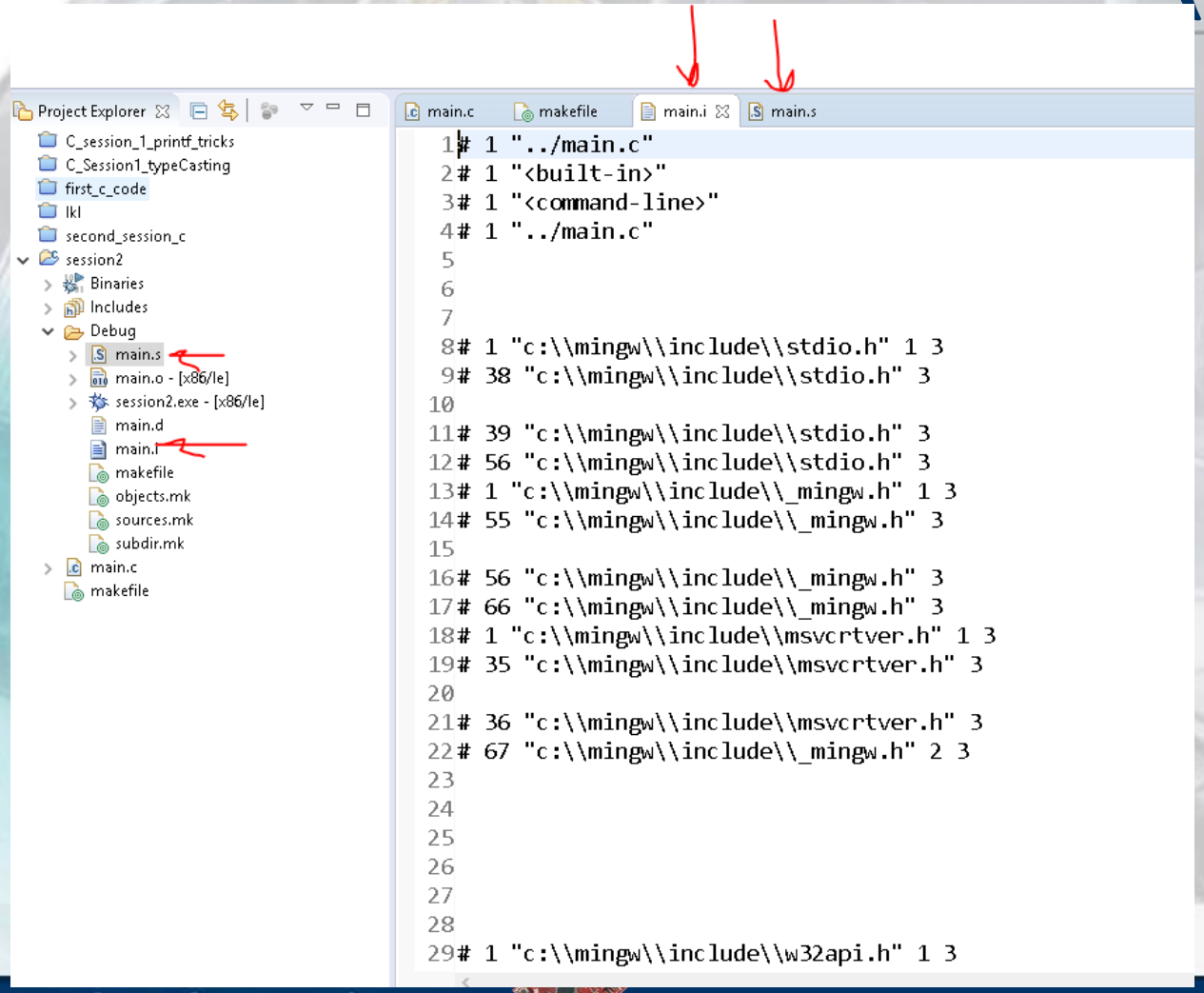
Macro
How To see the .i file (precompiled



Fourth : we will add a commands to generate .i and .s
By our hand on the Makefile

https://www.facebook.com/groups/embedded.system.KS/

Macro
How To see the .i file (precompiled

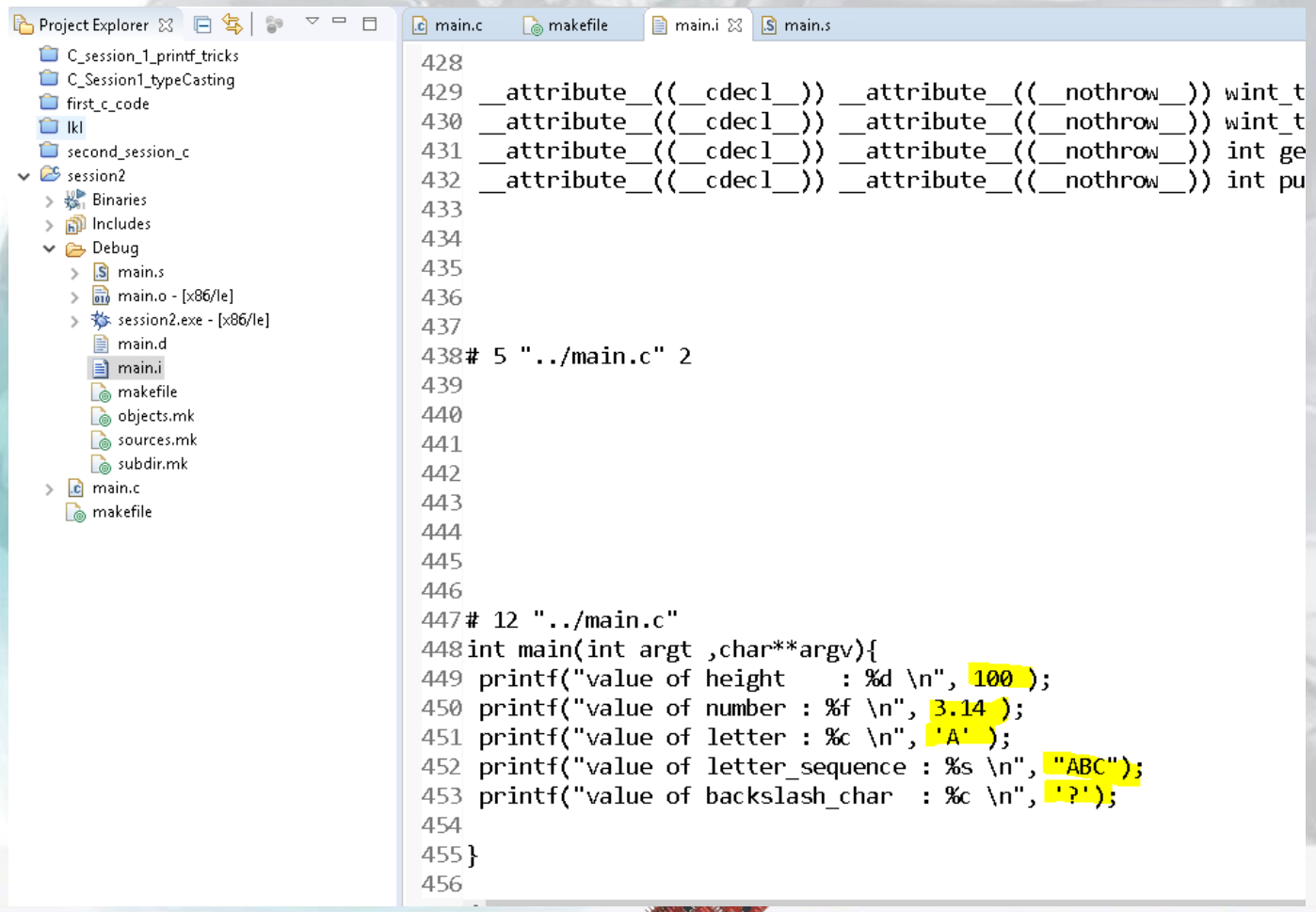


Fifth:
Rebuild again
You will see the files
Generated

We can see the precompiled file .i have a text replacement for MACRO

#LEARN_IN DEPTH

#Be_professional_in embedded_system



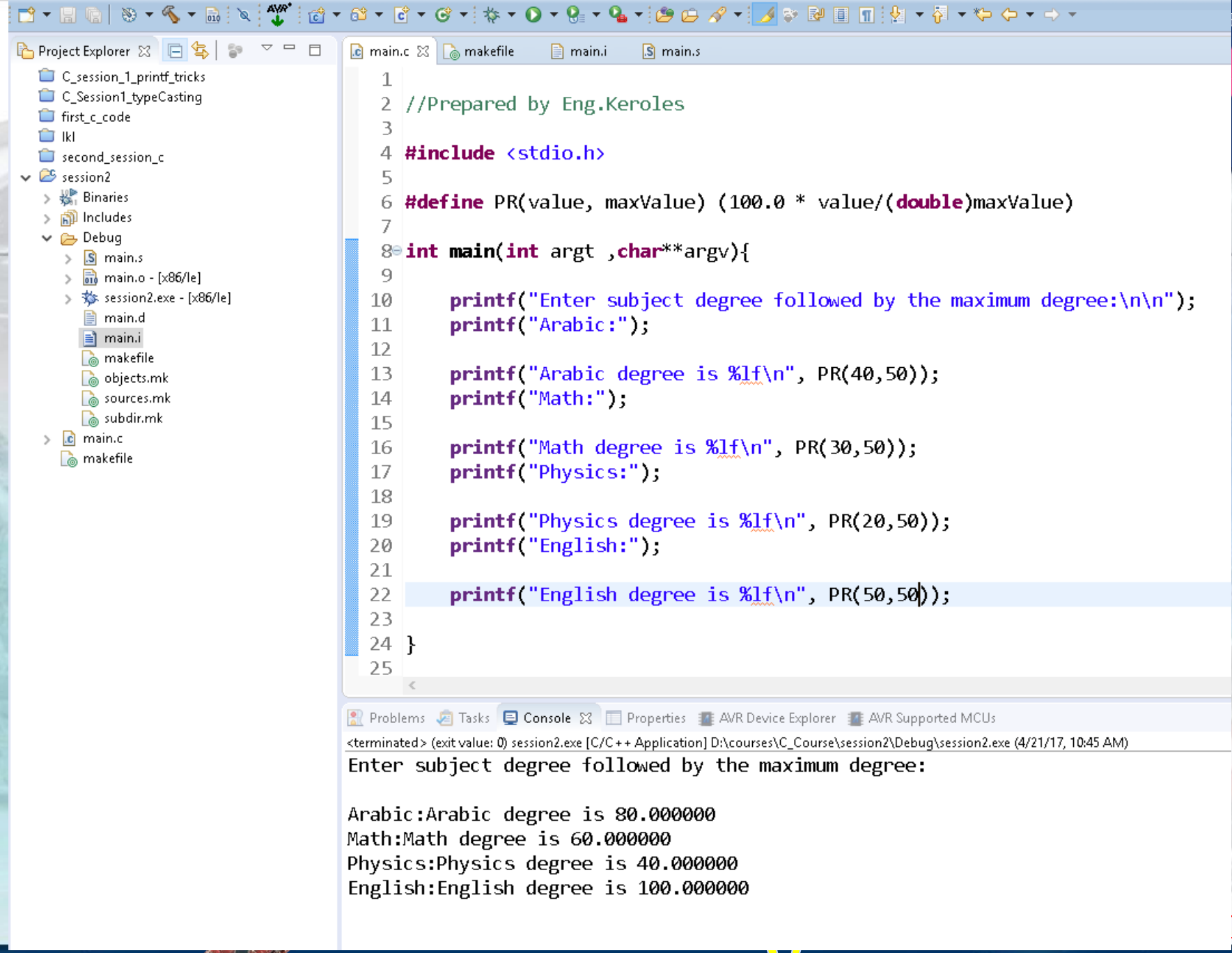
#define *identifier replacement-list*(optional)

Now are you agree with me

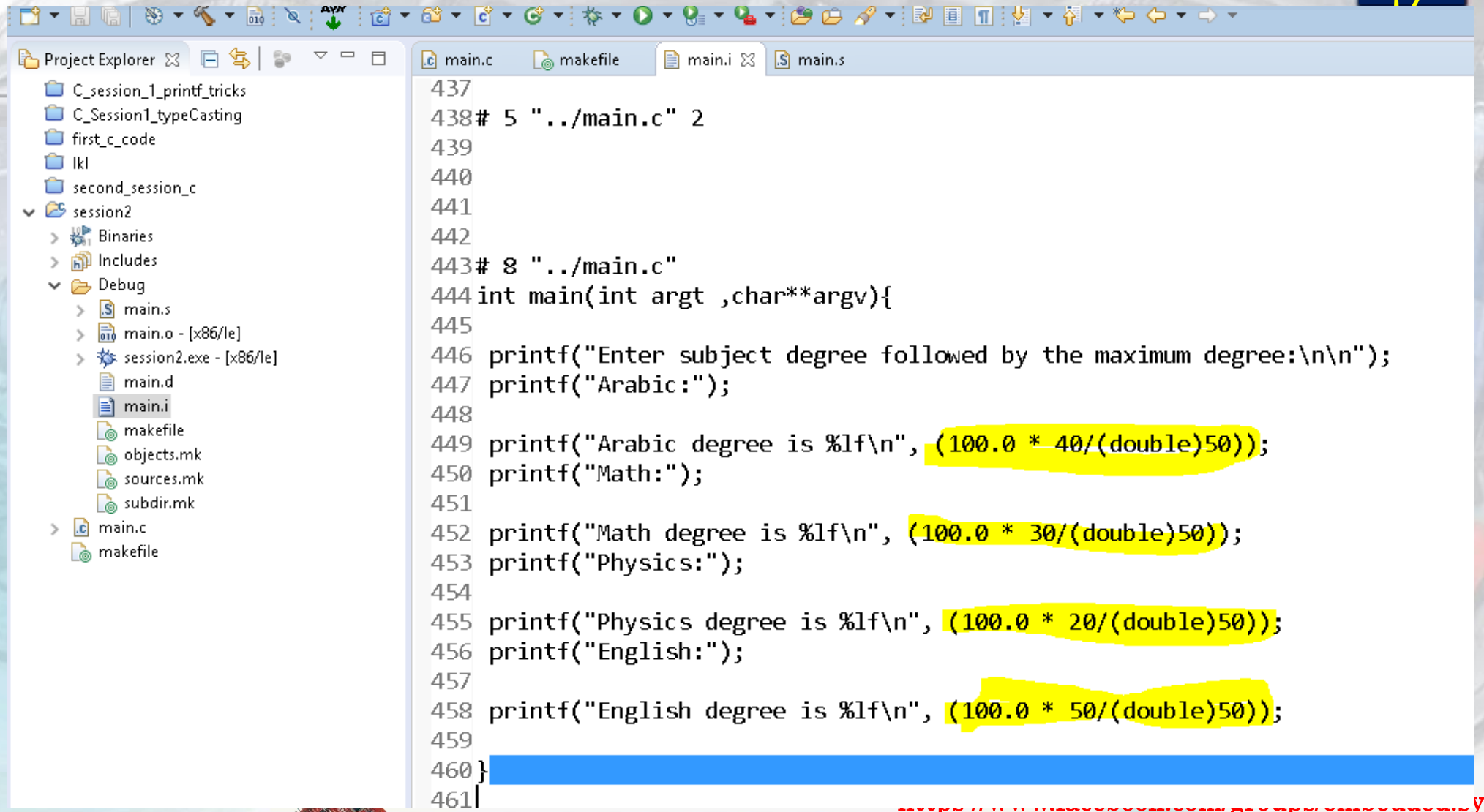
The #define directives define the identifier as a macro, that is they instruct the compiler to replace all successive occurrences of identifier with replacement-list

https://www.facebook.com/groups/embedded.system.KS/

Another Example



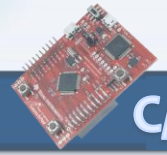
Another Example



__VA_ARGS__

`#define` *identifier*`(` *parameters*`, ... )` *replacement-list*	(3)
`#define` *identifier*`( ... )` *replacement-list*	(4)

- Version (3) of the #define directive defines a function-like macro with variable number of arguments. The additional arguments can be accessed using __VA_ARGS__ identifier, which is then replaced with arguments, supplied with the identifier to be replaced.
- Version (4) of the #define directive defines a function-like macro with variable number of arguments, but no regular arguments. The arguments can be accessed only with __VA_ARGS__ identifier, which is then replaced with arguments, supplied with identifier to be replaced.
- A ## operator between any two successive identifiers in the replacement-list runs parameter replacement on the two identifiers and then concatenates the result



Let us try

```c
//@ www.learn-in-depth.com
#include "stdio.h"
#define val 3
#define keroles(...) printf(__VA_ARGS__)
#define learn_in_depth(a,...) printf(__VA_ARGS__,a)

void main ()
{

printf ("value =%d \n", val);
keroles ("value =%d \n", val);
learn_in_depth (val, "value =%d \n");

}
```


Let us try

```c
//@ www.learn-in-depth.com
#include "stdio.h"
#define val 3
#define keroles(...) printf(__VA_ARGS__)
#define learn_in_depth(a,...) printf(__VA_ARGS__,a)

void main ()
{

printf ("value =%d \n", val);
keroles ("value =%d \n", val);
learn_in_depth (val, "value =%d \n");

}
```

```c
445
446
447  # 7 ".\\macro.c"
448  void main ()
449  {
450
451  printf ("value =%d \n", 3);
452  printf("value =%d \n", 3);
453  printf("value =%d \n",3);
454
455  }
456
```


#LEARN_IN DEPTH

#Be_professional_in_embedded_system

*main.c | makefile | main.i | main.s

```c
//Prepared by Eng.Keroles
#include <stdio.h>

#define CAT(a, ...) PRIMITIVE_CAT(a, __VA_ARGS__)
#define PRIMITIVE_CAT(a, ...) a ## __VA_ARGS__

#define DEC(x) PRIMITIVE_CAT(DEC_, x)
#define DEC_0 0
#define DEC_1 0
#define DEC_2 1
#define DEC_3 2
#define DEC_4 3
#define DEC_5 4
#define DEC_6 5
#define DEC_7 6
#define DEC_8 7
#define DEC_9 8

#define Dprintf(...) printf (__VA_ARGS__);

int main(int argt ,char**argv){

    Dprintf ("DEC(7)=%d",DEC(7));
}
```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 11:11 AM)

```
DEC(7)=6
```

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

*main.c | makefile | main.i | main.s

```c
//Prepared by Eng.Keroles
#include <stdio.h>

#define CAT(a, ...) PRIMITIVE_CAT(a, __VA_ARGS__)
#define PRIMITIVE_CAT(a, ...) a ## __VA_ARGS__

#define DEC(x) PRIMITIVE_CAT(DEC_, x)
#define DEC_0 0
#define DEC_1 0
#define DEC_2 1
#define DEC_3 2
#define DEC_4 3
#define DEC_5 4
#define DEC_6 5
#define DEC_7 6
#define DEC_8 7
#define DEC_9 8

#define Dprintf(...) printf (__VA_ARGS__);

int main(int argt ,char**argv){

    Dprintf ("DEC(7)=%d",DEC(7));
}
```

Project Explorer

- C_session_1_printf_tricks
- C_Session1_typeCasting
- first_c_code
- lkl
- second_session_c
- session2
  - Binaries
  - Includes
  - Debug
    - main.s
    - main.o - [x86/le]
    - session2.exe - [x86/le]
    - main.d
    - main.i
    - makefile
    - objects.mk
    - sources.mk
    - subdir.mk
  - main.c
  - makefile

main.c | makefile | main.i | main.s

```
434
435
436# 3 "../main.c" 2
437# 21 "../main.c"
438
439# 21 "../main.c"
440int main(int argt ,char**argv){
441
442 printf ("DEC(7)=%d",6);
443
444}
445
```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Sup

CDT Build Console [session2]

```
warning: MPFR header version 3.1.3 differs from libi
GGC heuristics: --param ggc-min-expand=100 --param
Compiler executable checksum: 7cd4969658477d55b298d
```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 11:11 AM)

```
DEC(7)=6
```

Think in Depth
how to overcome on this error without remove Dprintf from main

#LEARN_IN DEPTH

#Be_professional_in embedded_system

main.c | makefile | main.i | main.s

```
1  //Prepared by Eng.Keroles
2  #include <stdio.h>
3
4
5  #define Dprintf(...) printf (__VA_ARGS__)
6
7  int main(int argt ,char**argv){
8      Dprintf ("DEC(7)=%d",DEC(7));
9      Dprintf();
10 }
11
12
13
14
```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

CDT Build Console [session2]

```
    compiled by GNU C version 5.3.0, GMP version 6.1.0, MPFR version 3.1.3, MPC version 1.0.2
warning: GMP header version 6.1.0 differs from library version 5.1.2.
warning: MPFR header version 3.1.3 differs from library version 3.1.2.
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: 7cd4969658477d55b298dfed5a318eb7
../main.c: In function 'main':
../main.c:8:23: warning: implicit declaration of function 'DEC' [-Wimplicit-function-declaration]
  Dprintf ("DEC(7)=%d",DEC(7));
                       ^
../main.c:5:30: note: in definition of macro 'Dprintf'
 #define Dprintf(...) printf (__VA_ARGS__)
                              ^
../main.c:5:22: error: too few arguments to function 'printf'
 #define Dprintf(...) printf (__VA_ARGS__)
                      ^
../main.c:9:2: note: in expansion of macro 'Dprintf'
  Dprintf();
  ^
make: *** [main.o] Error 1

11:21:32 Build Finished (took 455ms)
```

Two errors

https://www.facebook.com/groups/embeddedsystem.KS/

Think in Depth
how to overcome on this error without remove Dprintf from main

#LEARN_IN DEPTH

#Be_professional_in embedded_system

main.c | makefile | main.i | main.s

```c
//Prepared by Eng.Keroles
#include <stdio.h>


#define Dprintf(...) printf (#__VA_ARGS__)

int main(int argt ,char**argv){
    Dprintf ("DEC(7)=%d",DEC(7));
    Dprintf();
}
```

Convert to string

Problems | Tasks | Console | Properties | AVR Device Exp

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Co
"DEC(7)=0",DEC(7)
```

main.c | makefile | main.i | main.s

```
434
435
436# 3 "../main.c" 2
437
438
439
440
441
442# 7 "../main.c"
443int main(int argt ,char**argv){
444 printf ("\"DEC(7)=%d\",DEC(7)");
445 printf ("");
446}
447
```

#undef directive

- The #undef directive undefines the identifier, that is it cancels the previous definition of the identifier by #define directive. If the identifier does not have an associated macro, the directive is ignored.

EXAMPLE PROGRAM FOR UNDEF IN C LANGUAGE:

main.c | makefile | main.i | main.s

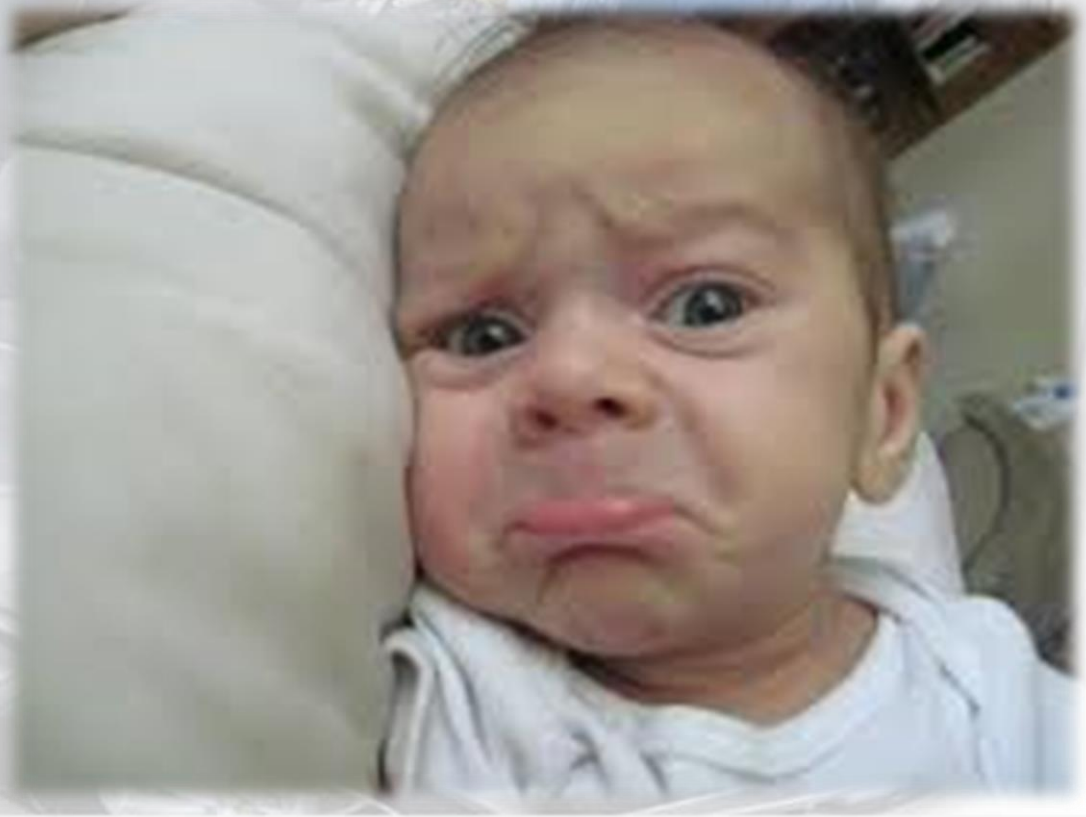
```c
//Prepared by Eng.Keroles
#include <stdio.h>

#define height 100

int main(int argt ,char**argv){
        printf("First defined value for height    : %d\n",height);
        #undef height           // undefining variable
        #define height 600      // redefining the same for new value
        printf("value of height after undef \& redefine:%d",height);

return 0 ;
}
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 12:16 PM)
First defined value for height    : 100
value of height after undef & redefine:600
```

Try to make function factory and use it by Macro

- Use one macro to create two Functions, one responsible on multiply input argument to 4 and the other to multiply one input argument to 2.
- The first function name will be fun_quadruple(int x)
- The Second one will be fun_double (int x)



main.c | makefile | main.i | main.s

```c
//Prepared by Eng.Keroles
#include <stdio.h>


//make function factory and use it
#define FUNCTION(name, a) int fun_##name(int x) { return (a)*x;}

FUNCTION(quadruple, 4)
FUNCTION(double, 2)

#undef FUNCTION
#define FUNCTION 34
#define OUTPUT(a) printf( #a )

int main(int argt ,char**argv){
     printf("quadruple(13): %d\n", fun_quadruple(13) );
        printf("double(21): %d\n", fun_double(21) );
        printf("%d\n", FUNCTION);
        OUTPUT(Keroles);    //convert million to string using #
}

```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 11:27 AM)

```
quadruple(13): 52
double(21): 42
34
Keroles
```

main.c | makefile | main.i | main.s

```
437
438
439
440
441
442
443 # 8 "../main.c"
444 int fun_quadruple(int x) { return (4)*x;}
445 int fun_double(int x) { return (2)*x;}
446
447
448
449
450
451 int main(int argt ,char**argv){
452   printf("quadruple(13): %d\n", fun_quadruple(13) );
453      printf("double(21): %d\n", fun_double(21) );
454      printf("%d\n", 34);
455      printf( "Keroles" );
456 }
457
```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 11:2

```
quadruple(13): 52
double(21): 42
34
Keroles
```

Predefined macros

The following macro names are predefined in any translation unit:

__STDC__	expands to the integer constant `1`. This macro is intended to indicate a conforming implementation (macro constant)
__STDC_VERSION__ (C95)	expands to an integer constant of type `long` whose value increases with each version of the C standard: • `199409L` (C95) • `199901L` (C99) • `201112L` (C11) (macro constant)
__STDC_HOSTED__ (C99)	expands to the integer constant `1` if the implementation is hosted (runs under an OS), `0` if freestanding (runs without an OS) (macro constant)
__FILE__	expands to the name of the current file, as a character string literal, can be changed by the #line directive (macro constant)
__LINE__	expands to the source file line number, an integer constant, can be changed by the #line directive (macro constant)
__DATE__	expands to the date of translation, a character string literal of the form "Mmm dd yyyy". The name of the month is as if generated by `asctime` and the first character of "dd" is a space if the day of the month is less than 10 (macro constant)
__TIME__	expands to the time of translation, a character string literal of the form "hh:mm:ss", as in the time generated by `asctime()` (macro constant)

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

main.c | makefile | main.i | main.s

```c
//Prepared by Eng.Keroles
#include <stdio.h>

int main(int argt ,char**argv){
      printf("__STDC__ = %ld  \n",__STDC__ );
      printf("__STDC_VERSION__ = %ld  \n",__STDC_VERSION__ );
      printf("__STDC_HOSTED__ = %ld  \n",__STDC_HOSTED__ );
      printf("__FILE__ = %s  \n",__FILE__ );
      printf("__LINE__ = %d  \n",__LINE__ );
      printf("__DATE__ = %s  \n",__DATE__ );
      printf("__TIME__ = %s  \n",__TIME__ );
      printf("__func__ = %s  \n",__func__ );

return 0 ;
}
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 11:54 AM)

```
__STDC__ = 1
__STDC_VERSION__ = 201112
__STDC_HOSTED__ = 1
__FILE__ = ../main.c
__LINE__ = 9
__DATE__ = Apr 21 2017
__TIME__ = 11:54:28
__func__ = main
```

Macros ==Text Replacement

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

__VA_ARGS__ & # &

```
#define Dprintf(...) printf (__VA_ARGS__);
```

Syntax

`#define` *identifier replacement-list* (optional)	(1)	
`#define` *identifier*`(` *parameters* `)` *replacement-list*	(2)	
`#define` *identifier*`(` *parameters*`, ... )` *replacement-list*	(3)	(since C99)
`#define` *identifier*`( ... )` *replacement-list*	(4)	(since C99)
`#undef` *identifier*	(5)	

main.c | makefile | main.i | main.s

```c
//Prepared by Eng.Keroles
#include <stdio.h>


//make function factory and use it
#define FUNCTION(name, a) int fun_##name(int x) { return (a)*x;}

FUNCTION(quadruple, 4)
FUNCTION(double, 2)

#undef FUNCTION
#define FUNCTION 34
#define OUTPUT(a) printf( #a )

int main(int argt ,char**argv){
     printf("quadruple(13): %d\n", fun_quadruple(13) );
        printf("double(21): %d\n", fun_double(21) );
        printf("%d\n", FUNCTION);
        OUTPUT(Keroles);    //convert million to string using #
}
```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 11:27 AM)

```
quadruple(13): 52
double(21): 42
34
Keroles
```

Predefined macros

main.c | makefile | main.i | main.s

```c
//Prepared by Eng.Keroles
#include <stdio.h>

int main(int argt ,char**argv){
     printf("__STDC__ = %ld  \n",__STDC__ );
     printf("__STDC_VERSION__ = %ld  \n",__STDC_VERSION__ );
     printf("__STDC_HOSTED__ = %ld  \n",__STDC_HOSTED__ );
     printf("__FILE__ = %s  \n",__FILE__ );
     printf("__LINE__ = %d  \n",__LINE__ );
     printf("__DATE__ = %s  \n",__DATE__ );
     printf("__TIME__ = %s  \n",__TIME__ );
     printf("__func__ = %s  \n",__func__ );

return 0 ;
}
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 11:54 AM)

```
__STDC__ = 1
__STDC_VERSION__ = 201112
__STDC_HOSTED__ = 1
__FILE__ = ../main.c
__LINE__ = 9
__DATE__ = Apr 21 2017
__TIME__ = 11:54:28
__func__ = main
```

Conditional compilation	**Syntax:** #ifdef, #endif, #if, #else, #ifndef Set of commands are included or excluded in source program before compilation with respect to the condition.

EXAMPLE PROGRAM FOR #IFDEF, #ELSE AND #ENDIF IN C:

- "#ifdef" directive checks whether particular macro is defined or not.
  If it is defined, "If" clause statements are included in source file.
- Otherwise, "else" clause statements are included in source file for compilation and execution.

EXAMPLE PROGRAM FOR #IFDEF, #ELSE AND #ENDIF IN C:

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

main.c | makefile | main.i | main.s

```c
//Prepared by Eng.Keroles
#include <stdio.h>

int main(int argt ,char**argv){
#ifdef Keroles_C_Course
printf("Keroles_C_Course is defined. So, this line will be added in " \
       "this C file\n");
#else
printf("Keroles_C_Course is not defined\n");
#endif

return 0 ;
}
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 12:08 PM)
Keroles_C_Course is not defined
```

main.c | makefile | main.i | main.s

```
438
439# 4 "../main.c"
440int main(int argt ,char**argv){
441
442
443
444
445printf("Keroles_C_Course is not defined\n");
446
447
448return 0 ;
449}
450
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.
Keroles_C_Course is not defined
```

EXAMPLE PROGRAM FOR #IFDEF, #ELSE AND #ENDIF IN C:

main.c | makefile | main.i | main.s

```c
//Prepared by Eng.Keroles
#include <stdio.h>
#define Keroles_C_Course
int main(int argt ,char**argv){
#ifdef Keroles_C_Course
printf("Keroles_C_Course is defined. So, this line will be added in " \
       "this C file\n");
#else
printf("Keroles_C_Course is not defined\n");
#endif

return 0 ;
}
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 12:09 PM)
Keroles_C_Course is defined. So, this line will be added in this C file
```

main.c | makefile | main.i | main.s

```
438
439# 4 "../main.c"
440int main(int argt ,char**argv){
441
442printf("Keroles_C_Course is defined. So, this line will be added in "
443       "this C file\n");
444
445
446
447
448return 0 ;
449}
450
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 12:09 PM)
Keroles_C_Course is defined. So, this line will be added in this C file
```

EXAMPLE PROGRAM FOR #IFNDEF AND #ENDIF IN C:

- **#ifndef** exactly acts as reverse as #ifdef directive. If particular macro is not defined, "If" clause statements are included in source file.
- Otherwise, else clause statements are included in source file for compilation and execution.

EXAMPLE PROGRAM FOR #IFNDEF AND #ENDIF IN C:

main.c | makefile | main.i | main.s

```
1  //Prepared by Eng.Keroles
2  #include <stdio.h>
3  //#define Keroles_C_Course
4  int main(int argt ,char**argv){
5  #ifndef Keroles_C_Course
6  printf("Keroles_C_Course is not defined\n");
7  #else
8  printf("Keroles_C_Course is defined. So, this line will be added in " \
9         "this C file\n");
10 #endif
11
12 return 0 ;
13 }
14
15
16
17
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 12:13 PM)
Keroles_C_Course is not defined
```

EXAMPLE PROGRAM FOR #IFNDEF AND #ENDIF IN C:

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

main.c | makefile | main.i | main.s

```
1  //Prepared by Eng.Keroles
2  #include <stdio.h>
3  #define Keroles_C_Course
4  int main(int argt ,char**argv){
5  #ifndef Keroles_C_Course
6  printf("Keroles_C_Course is not defined\n");
7  #else
8  printf("Keroles_C_Course is defined. So, this line will be added in " \
9         "this C file\n");
10 #endif
11
12 return 0 ;
13 }
14
15
16
17
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 12:13 PM)

```
Keroles_C_Course is defined. So, this line will be added in this C file
```


EXAMPLE PROGRAM FOR #IF, #ELSE AND #ENDIF IN C:

- **"If"** clause statement is included in source file if given condition is true.
- Otherwise, else clause statement is included in source file for compilation and execution.

```c
//Prepared by Eng.Keroles
#include <stdio.h>

#define a 100

int main(int argt ,char**argv){
#if (a==100)
  printf("This line will be added in this C file since " \
         "a \= 100\n");
  #else
  printf("This line will be added in this C file since " \
         "a is not equal to 100\n");
  #endif

return 0 ;
}
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 12:15 PM)

```
This line will be added in this C file since a = 100
```

Quiz: Think how to write a code can be Removed or added to the Embedded C Program and Responsible on print Debug with Details

Create Debug messages with level details

```
Windows PowerShell
PS C:\MinGW\bin> .\macro.exe
value =3
value =3
value =3
>>      helllo
@ Func: main, File: .\macro.c, Line: 29  >>     init the CAN Controller
PS C:\MinGW\bin>
```

vel details

```c
//@ www.learn-in-depth.com
#include "stdio.h"
#define val 3
#define keroles(...) printf(__VA_ARGS__)
#define learn_in_depth(a,...) printf(__VA_ARGS__,a)

#define debug

#ifdef debug
int debug_enable = 1 ;
#else
int debug_enable = 0 ;
#endif

#define DPRINTF(debug_level,...)   if (debug_enable)  { \
                        if (debug_level > 1) printf ("@ Func: %s, File: %s, Line: %d  ",__func__, __FILE__, __LINE__); \
                        printf (">> \t");\
                        printf (__VA_ARGS__);}

void main ()
{

printf ("value =%d \n", val);
keroles ("value =%d \n", val);
learn_in_depth (val, "value =%d \n");

DPRINTF (1,"helllo \n");

DPRINTF(3, "init the CAN Controller  \n");


}
```

macro.c — C:\MinGW\bin | macro.i | macro_lab.txt

```
449
450  # 10 ".\\macro.c"
451  int debug_enable = 1 ;
452  # 20 ".\\macro.c"
453  void main ()
454  {
455
456  printf ("value =%d \n", 3);
457  printf("value =%d \n", 3);
458  printf("value =%d \n",3);
459
460  if (debug_enable) { if (1 > 1) printf ("@ Func: %s, File: %s, Line: %d  ",__func__, ".\\macro.c", 27); printf (">> \t"); printf ("helllo
     \n");};
461
462  if (debug_enable) { if (3 > 1) printf ("@ Func: %s, File: %s, Line: %d  ",__func__, ".\\macro.c", 29); printf (">> \t"); printf ("init the
     CAN Controller  \n");};
463
464
465  }
466
```

Pragma directive in c

- Pragma is **implementation specific directive** i.e **each pragma directive has different implementation rule and use** .
- There are **many** type of pragma directive and **varies from one compiler to another compiler** .
- **If compiler does not recognize particular pragma the it simply ignore that pragma statement without showing any error message and execute the whole program assuming this pragma statement is not present.**

Pragma directive in c

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

main.c | makefile | main.i | main.s

```
1  //Prepared by Eng.Keroles
2  #include <stdio.h>
3
4  #pragma world
5
6  int main(int argt ,char**argv){
7      printf ( "\n Now we are in main function" ) ;
8      return 0  ;
9  }
10
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

CDT Build Console [session2]

```
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: 7cd4969658477d55b298dfed5a318eb7
../main.c:4:0: warning: ignoring #pragma world  [-Wunknown-pragmas]
 #pragma world
 ^
COLLECT_GCC_OPTIONS='-O0' '-g3' '-Wall' '-c' '-fmessage-length=0' '-v' '-MMD' '-MP' '-MF' 'main.d' '-MT' 'main.o
 c:/mingw/bin/../lib/gcc/mingw32/5.3.0/../../../../mingw32/bin/as.exe -v -o main.o C:\Users\kkhalil\AppData\Local
GNU assembler version 2.25.1 (mingw32) using BFD version (GNU Binutils) 2.25.1
COMPILER_PATH=c:/mingw/bin/../libexec/gcc/mingw32/5.3.0/;c:/mingw/bin/../libexec/gcc/;c:/mingw/bin/../lib/gcc/mi
LIBRARY_PATH=c:/mingw/bin/../lib/gcc/mingw32/5.3.0/;c:/mingw/bin/../lib/gcc/;c:/mingw/bin/../lib/gcc/mingw32/5.3
COLLECT_GCC_OPTIONS='-O0' '-g3' '-Wall' '-c' '-fmessage-length=0' '-v' '-MMD' '-MP' '-MF' 'main.d' '-MT' 'main.o
Finished building: ../main.c

Keroles
Building target: session2.exe
Invoking: MinGW C Linker
Keroles: Generate PrePcomiler file .i
gcc -E ../main.c  -o ./main.i
Keroles: Generate Assembly file .s
gcc -S ../main.c  -o ./main.s
gcc  -o "session2.exe"  ./main.o
Finished building target: session2.exe


12:30:32 Build Finished (took 1s.149ms)
```

Problems | AVR Supported MCUs | Tasks

<terminated> (exit value: 0) session2.exe [C/C++ Appli

```
Now we are in main function
```

https://www.facebook.com/groups/embedded.system.KS/

We will take

- #pragma startup
- #pragma exit

What is #pragma startup and #pragma exit in c programming language?

- #pragma startup [priority]
- #pragma exit [priority]

- Where priority is optional integral number.
- For user priority varies from 64 to 255
- For c libraries priority varies from 0 to 63
- Default priority is 100.

What is #pragma startup and #pragma exit in c programming language?

- **pragma startup** always execute the function **before the main** function

  pragma exit always execute the function **after the main** function.Function declaration of must be before startup and exit pragma directives and function must not take any argument and return void. If more than one startup directive then priority decides which will execute first.

- startup:
  - Lower value: higher priority i.e. functions will execute first. If more than one exit directive then priority decides which will execute first.
- exit:
  - Higher value: higher priority i.e. functions will execute first. For example


```
void india();
void usa() ;
#pragma startup india 105
#pragma startup usa
#pragma exit usa
#pragma exit india 105

void main(){
    printf("\nI am in main");
    getch();
}

void india(){
    printf("\nI am in india");
    getch();
}

void usa(){
    printf("\nI am in usa");
    getch();
}
```

Output:

```
I am in usa
I am in India
I am in main
I am in India
I am in usa
```

#pragma startup/exit ... is not supported in gcc.

- In gcc and clang, you should use **__attribute__((constructor))** and **__attribute__((destructor))** instead.

main.c | makefile | main.i | main.s

```
1  //Prepared by Eng.Keroles
2  #include <stdio.h>
3
4  void india();
5  void usa() ;
6  void __attribute__((constructor)) india();
7  void __attribute__((destructor)) usa();
8
9
10 int main(int argt ,char**argv) {
11     printf("\nI am in main") ;
12     return 0   ;
13 }
14
15 void india(){
16     printf("\nI am in india");
17 }
18
19 void usa(){
20     printf("\nI am in usa");
21 }
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/21/17, 12:41 PM)

```
I am in india
I am in main
I am in usa
```

Macros Vs Functions

No	Macro	Function
1	Macro is **Preprocessed**	Function is **Compiled**
2	**No Type Checking**	**Type Checking** is Done
3	**Code** Length **Increases**	**Code** Length remains **Same**
–		
5	Speed of Execution is **Faster**	Speed of Execution is **Slower**
6	Before Compilation macro name is replaced by macro value	During function call , Transfer of Control takes place
7	Useful where small code appears many time	Useful where large code appears many time
8	Generally Macros do not extend beyond one line	Function can be of any number of lines

ANSI C

- **ANSI C, ISO C** and **Standard C** refer to the successive standards for the C programming language published by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). Historically, the names referred specifically to the original and best-supported version of the standard (known as **C89** or **C90**). Software developers writing in C are encouraged to conform to the standards, as doing so aids portability between compilers.
- C89
- C90
- C95
- C99

Preprocessor Test for C95 compatibility

```
#if defined(__STDC_VERSION__) && __STDC_VERSION__ >= 199409L

/* C95 compatible source code. */
#elif defined(__ANSI__)
/* C89 compatible source code. */
#endif
```

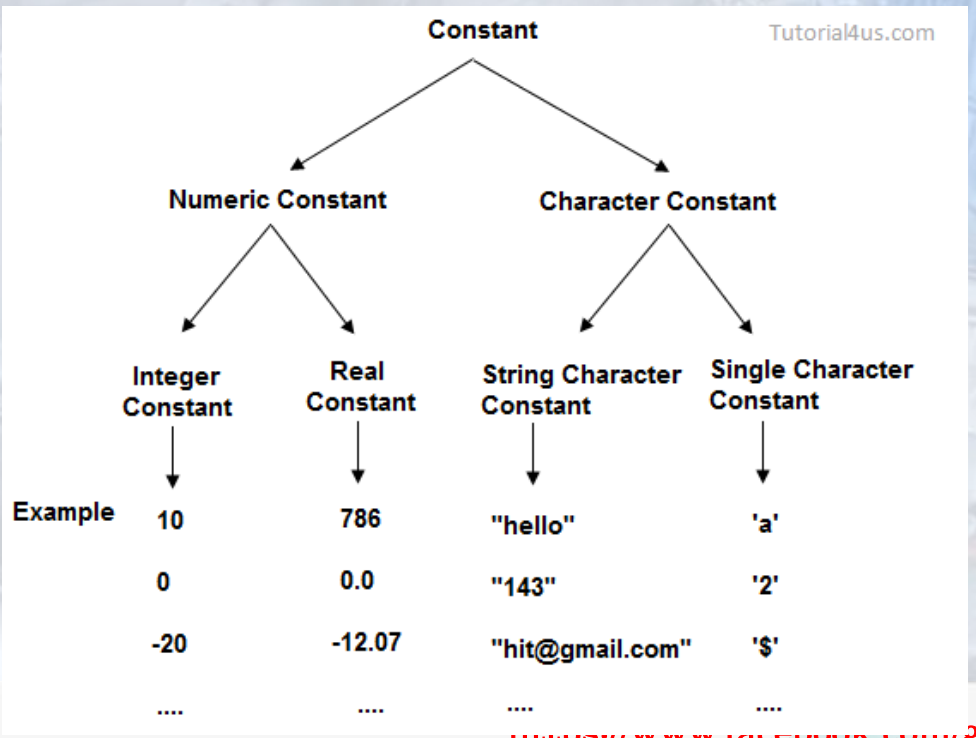
Type qualifier in C

- There are two important type qualifiers in C :
  - const
  - volatile
- Applying these qualifiers to variable declaration is called qualifying the declaration. The qualifier allows the programmer to add some features to a variable.
- The **const** type qualifier in C is used to **enforce read only features** on variables

Constant in C

- **constant** can be used to represent as fixed values in a C program
- Constants refers to the fixed values that do not change during the execution of a program

```
1.const float pie =3.147;
2.const int radius =4;
3.const char c = 'A';
4.const char name[] = “C Course";
```



Constant using const keyword C programming:

- When declaring a const variable, it is possible to put const either before or after the type: that is, both

```
int const a = 15;
```

Or

```
const int X = 15;
```

Declare constant

- **const** keyword are used for declare a constant.

main.s | main.c

```
1  //Prepared by Eng.Keroles
2  #include <stdio.h>
3
4
5  int main(int argt ,char**argv) {
6      const int kkk=10;
7      printf("%d",kkk);
8          kkk=20; // gives error you can't modify const
9      return 0
10 }
```

gives error
you can't modify const

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

CDT Build Console [session2]

```
GNU C11 (GCC) version 5.3.0 (mingw32)
    compiled by GNU C version 5.3.0, GMP version 6.1.0, MPFR version 3.1.3, MPC ver
warning: GMP header version 6.1.0 differs from library version 5.1.2.
warning: MPFR header version 3.1.3 differs from library version 3.1.2.
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: 7cd4969658477d55b298dfed5a318eb7
../main.c: In function 'main':
../main.c:8:6: error: assignment of read-only variable 'kkk'
   kkk=20; // gives error you can't modify const
      ^
make: *** [main.o] Error 1

21:57:36 Build Finished (took 494ms)
```

Interview trick ☺
how to modify it.

Interview trick ☺ how to modify it.

- You can still modify the content of the variable by using its address

```c
//@ www.learn-in-depth.com
#include "stdio.h"


void main ()
{
    const int x = 3 ;
    printf ("x =%d \n",x);
    int* px= &x ;
    *px = 4 ;
    printf ("x =%d \n",x);



}
```

Expect the output ☺

Interview trick ☺ how to modify it.

- You can still modify the content of the variable by using its address

```c
//@ www.learn-in-depth.com
#include "stdio.h"


void main ()
{
    const int x = 3 ;
    printf ("x =%d \n",x);
    int* px= &x ;
    *px = 4 ;
    printf ("x =%d \n",x);



}
```

Expect the output ☺

```
Windows PowerShell
PS C:\MinGW\bin> .\constant.exe
x =3
x =4
PS C:\MinGW\bin>
```

Another example

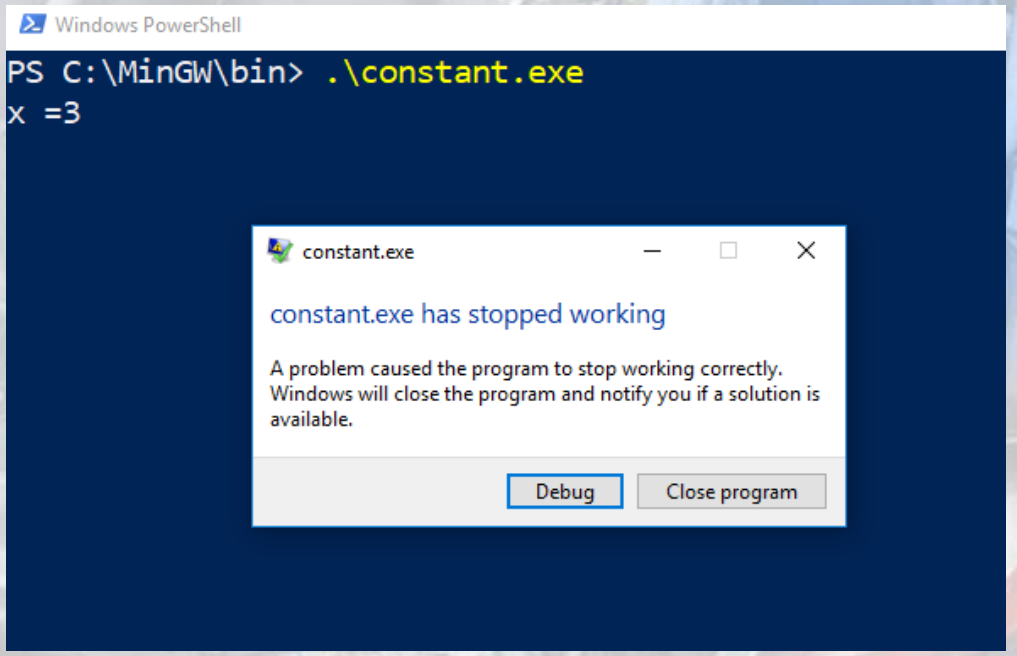
```c
//@ www.learn-in-depth.com
#include "stdio.h"

    const int x = 3 ;

void main ()
{
    printf ("x =%d \n",x);
    int* px= &x ;
    *px = 4 ;
    printf ("x =%d \n",x);


}
```

Expect the output ☺

Another example

```c
//@ www.learn-in-depth.com
#include "stdio.h"

    const int x = 3 ;

void main ()
{
    printf ("x =%d \n",x);
    int* px= &x ;
    *px = 4 ;
    printf ("x =%d \n",x);


}
```

Expect the output ☺



Learn in depth the constant

- **const** doesn't mean that **the value never** changes, its only programming **safety feature** to ensure that the programmer shouldn't try to modify the value.
- **All const variables** are **stored in memory in the same way as standard variables**.
- They are placed in **RAM**. The only difference, in this sense, is **the read-only feature.**
- **All global const** variables are stored in **ROM** or **FLASH memory**, This also further depends on linker script rules and the hardware on which code runs.
  - The flash memory of the micro-controller is indeed **write-protected, that means** the operation won't have any effect
  - **In the PC,** the program crashes because we're trying to write in the write protected section

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

main.s | main.c

```c
//Prepared by Eng.Keroles
#include <stdio.h>


int main(int argt ,char**argv) {
    const int kkk=10;
    printf("%d",kkk);
    return 0   ;
}
```

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/27/17, 9:

```
10
```

https://www.facebook.com/groups/embedded.system.KS/