C Source File Structure

- File Header
- Preprocessor statements
- Include files
- External declarations – (should be in a header file)
- Initialization of global data
- Local function definitions
- Main function

File Header

```
/*********************************************************************************
 **    FILENAME       :   ss_esl_refer_ext.c
 **
 **    DESCRIPTION    :   This file defines the functions which send
 **                       external messages for Refer.
 **
 **    Revision History    :
 **    DATE        NAME                REFERENCE         REASON
 **    -------------------------------------------------------
 **    11 Sept 2002       Mayank Rastogi          SPR 1204    New code for RY feature
**              31 May 2012          Vikas Nagpal              SPR 1211    Fixed the cause of unhappiness
**                                                                        by commenting out complaints
**

 **    Copyright © 2019 Altran  Group  All Rights Reserved
 **
 *********************************************************************************/
```

Include File Structure

- File header
- Embedded include file references
- #defines, macros
- Type definitions
- “extern” variable declarations
- “extern” function declarations

Example (Header File)

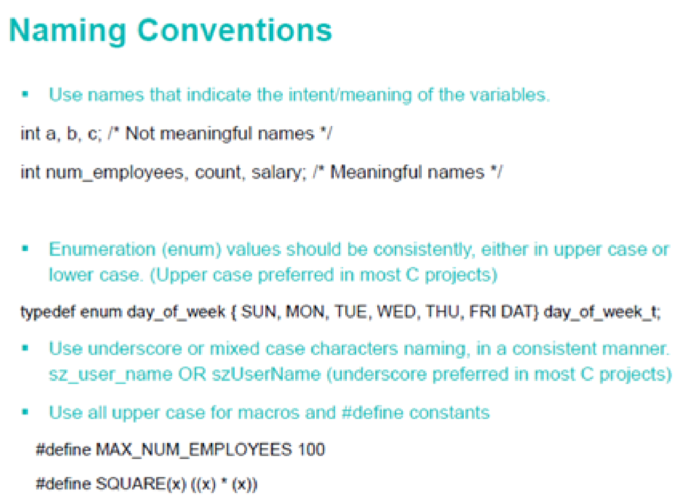
Technique for preventing multiple inclusion of include file

```
#ifndef   __SS_ESL_REFER_MACRO_H

#define  __SS_ESL_REFER_MACRO_H

/* Notice that the macro name is derived from the file name itself. That makes it unique. Underscores can be
added to the macro name to avoid accidental clash with some other name.

*/

#include "ss_esl_traces.h"

#include "ss_esl_macros.h"

#define ESL_REFER_INVALID_ARG 0


#ifdef ESL_TRACE_ENABLED

#define ESL_REFER_TRACE(trc_id, no_int_args, arg1, arg2, arg3, arg4) {

/*Write definition here*/}

#else

#define ESL_REFER_TRACE(trc_id, no_int_args, arg1, arg2, arg3, arg4)

#endif /* end of ifdef ESL_TRACE_ENABLED */

#endif /* End of ifndef __SS_ESL_REFER_MACRO_H */
```

Function Header

```
/*******************************************************

**FUNCTION NAME    :   ss_esl_refer_arm

**DESCRIPTION      :   In this function sends a call event

**                     request to SF for arming the specified event on the

**                     specified leg.

**

**RETURNS          :  ESL_SUCCESS, and in case of any error

*                     ESL_FAILURE with the corresponding ecode.

**

**********************************************************/

return_t ss_esl_refer_arm(

   U8 *p_msg; /* Pointer to message (Notice : argument meaning can be specified adjacent to its declaration ) */

   U16 len;) /* Length of message */

{

/* Function body */

}
```

Single and Multiline comments format and usage with example



Naming Conventions

- Use names that indicate the intent/meaning of the variables.

```
int a, b, c; /* Not meaningful names */

int num_employees, count, salary; /* Meaningful names */
```

- Enumeration (enum) values should be consistently, either in upper case or lower case. (Upper case preferred in most C projects)

```
typedef enum day_of_week { SUN, MON, TUE, WED, THU, FRI DAT} day_of_week_t;
```

- Use underscore or mixed case characters naming, in a consistent manner. sz_user_name OR szUserName (underscore preferred in most C projects)
- Use all upper case for macros and #define constants

```
  #define MAX_NUM_EMPLOYEES 100
  #define SQUARE(x) ((x) * (x))
```

Naming Conventions

- Do NOT use names that differ only by the case of characters. e.g. `SzName` and `szName`
- Do NOT start names with "_" or "__".
- Do NOT rename operations using macros. Example

```
#define EQ ==        /* avoid */
```

The reason why people like to define EQ is to avoid coding mistakes like **`if(A=B).`** But there are better ways to detect this problem. e.g **a)** Compile with –Wall **b)** Keep the constant to the left hand side of comparison

Functions and Files

- Structures should not be passed to functions because the whole structure is pushed onto stack. Use pointers instead
- Place machine-dependent code in a special file so that it may be easily located when porting code from one machine to another
- Use “const” to specify that the argument is un-modified by the function

Example : char *strcpy(char *dest, const char *src);

ACG Cheatsheet

General:

- File description. (.c, .h files)
- Function description (including main())
- For .h file use -#ifndef <macro_name> and a #define <macro_name>
- Declare 1 variable/line
- No magic no.'s. Use macros instead.
- Use library functions if it exists.
- Add Inline comment for a complex logical block.
- Add New line after each logical block of code. (declaration, initialization , processing etc)
- Enum values in upper case.
- Indent block code by 1 tab. Uniform spacing
- No/Minimal use of global data.
- Avoid goto.
- Add extern declarations in .h file.
- Use void for no argument.
- Name every formal arguments both in declaration and definition (.h, .c).
- Do not keep one function/file, rather keep related functions for a task in one file
- Do not keep implementations of helper functions (i.e of functions called from within main()) in the file having main(). Keep them in a separate file
- Include user defined includes in <> (not like #include "myutil.h")
- Follow correct the order for macros, #include, extern, global, local declarations as in ACG
- Use << and >> for multiplication and division.
- Prefer to avoid nested conditional expressions. viz-> ? : ? :
- Prefer not to use -- and ++ in logical expressions
- Include/exclude platform specific code via macros
- In case of switch cases, use macro's for case value
- Avoid double free
- Do not declare array with dimension as run time variables
- Mark end of loop for easy identification. (only if large no of lines in loop). (Optional).

Pointer, char initialization:

- use static with integers
- use NULL only with pointers.
- use '\0' for only ASCII characters.
- use (void *) for pointers of untyped/generic data types.

Data Structures:

- Choose appropriate data type and data structure.
- Use typedef for structure, union, enum with proper names.

Functions:

- All functions with the return type (Use SUCCESS /FAILURE or other values). Return value to be used by caller.
- Check every system and library call for errors if any and handle properly.
- Handle errors after malloc.
- Do not use arguments without validating argc
- In case of failure/success exit, ensure that cleanup is done on exit (i.e close open files, release allocated heap memory, close db connection etc).
- Put function arguments one/line indented 1 level.
- Multi word variable names using _ or camel notation.
- Single line comment to the right of the statement if fits in the same line. (max line length 80).
- Function should not be too lengthy (i.e shouldn't run into multiple screens).

Horizontal space:

- after every keyword for eg space after #include (i.e #include <stdio.h>).
- On both sides of the binary operator
- after every function argument.

Macros:

- Define macro in upper case
- Use parenthesis for macro arguments.
- For multiline macro, use space followed by \ after each line
- Include appropriate macros (for SUCCESS and FAILURE etc, but do not overdo like #define ONE 1)