# C programming

Trainer : Nisha Dingare

Email : nisha.dingare@sunbeaminfo.com

# Preprocessor Directives :

- Preprocessor is part of C programming toolchain/SDK.
  - Removes comments from the source code.
  - Expand source code by processing all statements starting with #.
  - Executed before compiler
- All statements starting with # are called as preprocessor directives.
  - Header file include
    - #include
  - Symbolic constants & Macros
    - #define
  - Conditional compilation
    - #if, #else, #elif, #endif
    - #ifdef #ifndef
  - Miscellaneous
    - #pragma, #error

# Preprocessor Directives :

- Preprocessor is small application helps to generate intermediate code.
- It process all c statements starting with # which is called as preprocessor directives.
- Preprocessor directives are the commands given preprocessor about processing source code before compilation.

- can be used for defining symbolic constant or macro
- #define <symbol> <replacable text>

# #include :

- #include includes header files (.h) in the source code (.c).
- #include <file.h>
  - Find file in standard include directory.
  - If not found, raise error.
- #include "file.h"
  - File file in current source directory.
  - If not found, find file in standard include directory.
  - If not found, raise error.

# #define symbolic constants :

- Used to define symbolic constants.
  - #define PI 3.142
  - #define SIZE 10
- Predefined constants
  - __LINE__
  - __FILE__
  - __DATE__
  - __TIME__
- Symbolic constants and macros are available from there declaration till the end of file. Their scope is not limited to the function.

# #define macros :

- Used to define macros (with or without arguments)
  - #define ADD(a, b) (a + b)
  - #define SQUARE(x) ((x) * (x))
  - #define SWAP(a,b,type) { type t = a; a = b; b = t; }
- Macros are replaced with macro expansion by preprocessor directly.
  - May raise logical/compiler errors if not used parenthesis properly.
- Stringizing operator (#)
  - Converts given argument into string.
  - #define PRINT(var) printf(#var " = %d", var)
- Token pasting operator (##)
  - Combines argument(s) of macro with some symbol.
  - #define VAR(a,b) a##b

# Difference between functions and macros :

- Functions
  - Function have declaration, definition and call.
  - Functions are called at runtime by creating FAR on stack.
  - Functions are type-safe.
  - Functions may be recursive.
  - Functions called multiple times doesn't increase code size.
  - Functions execute slower.
  - For bigger reusable code snippets, functions are preferred.
- Macros
  - Macro definition contain macro arguments and expansion.
  - Macros are replaced blindly by the processor before compilation
  - Macros are not type-safe.
  - Macros cannot be recursive.
  - Macros (multi-line) called multiple times increase code size.
  - Macros execute faster.
  - For smaller code snippets/formulas, macros are preferred.

# Conditional Compilation :

- As preprocessing is done before compilation, it can be used to control the source code to be made available for compilation process.
- The condition should be evaluated at preprocessing time (constant values).
- Conditional compilation directives
  - #if, #elif, #else, #endif
  - #ifdef, #ifndef

- #if
- #else
- #elif
- #endif
- #ifdef
- #ifndef
- Example:
- #define PI 3.14 ,
- #if defined (PI)
- printf("DEFINED");
- #else
- printf("NOT DEFINED");
- #endif

# Thank You !!