C - PREPROCESSORS

Dr. Sheak Rashed Haider Noori
Professor & Associate Head
Department of Computer Science

C - PREPROCESSORS

- 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.
- It must be the first nonblank character, and for readability.
- Below is the list of preprocessor directives that C language offers.

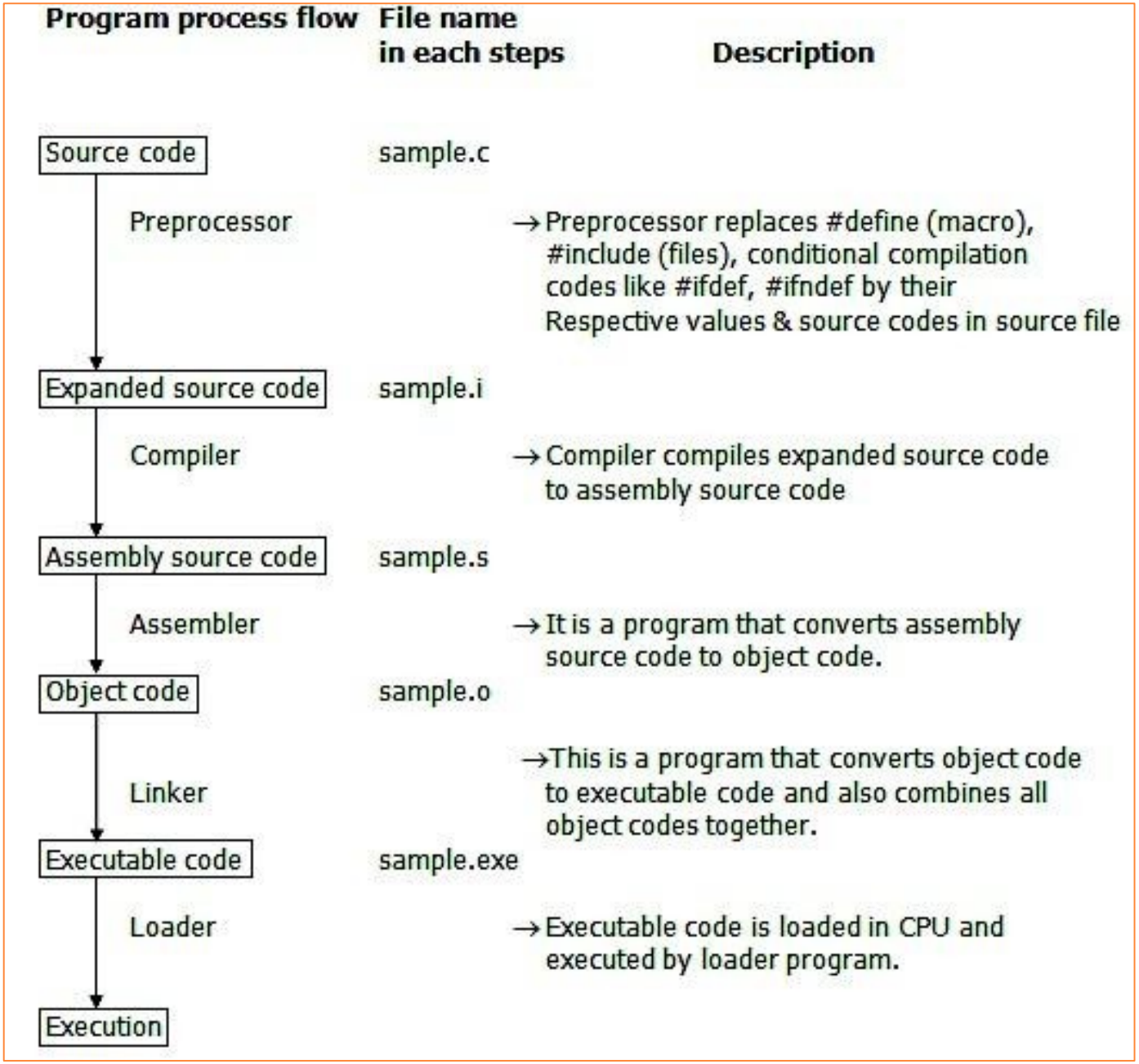
S.no	Preprocessor	Syntax	Description
1	Macro	#define	This macro defines constant value and can be any of the basic data types.
2	Header file inclusion	#include <file_name>	The source code of the file "file_name" is included in the main program at the specified place
3	Conditional compilation	#ifdef, #endif, #if, #else, #ifndef	Set of commands are included or excluded in source program before compilation with respect to the condition
4	Other directives	#undef, #pragma	#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

IMPORTANT PREPROCESSOR DIRECTIVES

Following section lists down all important preprocessor directives:

Directive	Description
#define	Substitutes a preprocessor macro
#include	Inserts a particular header from another file
#undef	Undefines a preprocessor macro
#ifdef	Returns true if this macro is defined
#ifndef	Returns true if this macro is not defined
#if	Tests if a compile time condition is true
#else	The alternative for #if
#elif	#else an #if in one statement
#endif	Ends preprocessor conditional
#error	Prints error message on stderr
#pragma	Issues special commands to the compiler, using a standardized method

A program in C language involves into different processes. Below diagram will help you to understand all the processes that a C program comes across.



PREPROCESSORS EXAMPLES

```
#define MAX_ARRAY_LENGTH 20
```

This directive tells the CPP to replace instances of MAX_ARRAY_LENGTH with 20. Use *#define* for constants to increase readability.

```
#include <stdio.h>
#include "myheader.h"
```

These directives tell the CPP to get stdio.h from **System Libraries** and add the text to the current source file. The next line tells CPP to get **myheader.h** from the local directory and add the content to the current source file.

```
#undef  FILE_SIZE
#define FILE_SIZE 42
```

This tells the CPP to undefine existing FILE_SIZE and define it as 42.

```
#ifndef MESSAGE
   #define MESSAGE "You wish!"
#endif
```

This tells the CPP to define MESSAGE only if MESSAGE isn't already defined.

```
#ifdef DEBUG
   /* Your debugging statements here */
#endif
```

This tells the CPP to do the process the statements enclosed if DEBUG is defined. This is useful if you pass the *-DDEBUG* flag to gcc compiler at the time of compilation. This will define DEBUG, so you can turn debugging on and off on the fly during compilation.

PREDEFINED MACROS

ANSI C defines a number of macros. Although each one is available for your use in programming, the predefined macros should not be directly modified.

Macro	Description
__DATE__	The current date as a character literal in "MMM DD YYYY" format
__TIME__	The current time as a character literal in "HH:MM:SS" format
__FILE__	This contains the current filename as a string literal.
__LINE__	This contains the current line number as a decimal constant.
__STDC__	Defined as 1 when the compiler complies with the ANSI standard.

```c
#include <stdio.h>

main()
{
   printf("File :%s\n", __FILE__ );
   printf("Date :%s\n", __DATE__ );
   printf("Time :%s\n", __TIME__ );
   printf("Line :%d\n", __LINE__ );
   printf("ANSI :%d\n", __STDC__ );

}
```

Output

```
File :test.c
Date :Jun 2 2012
Time :03:36:24
Line :8
ANSI :1
```

EXAMPLE PROGRAM FOR #DEFINE, #INCLUDE PREPROCESSORS IN C

- #define - This macro defines constant value and can be any of the basic data types.
- #include <file_name> - The source code of the file "file_name" is included in the main C program where "#include <file_name>" is mentioned.

```c
#include <stdio.h>

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

void main()
{
   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);

}
```

Output

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

EXAMPLE PROGRAM FOR #IFDEF, #ELSE AND #ENDIF

- "#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.

```
#include <stdio.h>
#define RAJU 100

int main()
{
   #ifdef RAJU
   printf("RAJU is defined. So, this line will be added in this C file\n");
   #else
   printf("RAJU is not defined\n");
   #endif
   return 0;
}
```

Output
RAJU is defined. So, this line will be added in this C file

EXAMPLE PROGRAM FOR #IFNDEF AND #ENDIF

- #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.

```
#include <stdio.h>
#define RAJU 100

int main()
{
   #ifndef SELVA
   {
      printf("SELVA is not defined. So, now we are going to define here\n");
      #define SELVA 300
   }
   #else
   printf("SELVA is already defined in the program");

   #endif
   return 0;
}
```

Output

SELVA is not defined. So, now we are going to define here

EXAMPLE PROGRAM FOR #IF, #ELSE AND #ENDIF

- "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.

```
#include <stdio.h>
#define a 100

int main()
{
   #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;
}
```

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

EXAMPLE PROGRAM FOR UNDEF

- This directive undefines existing macro in the program.

```
#include <stdio.h>

#define height 100
void main()
{
   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);
}
```

Output
First defined value for height : 100
value of height after undef & redefine:600

EXAMPLE PROGRAM FOR PRAGMAM

```
#include <stdio.h>

void function1( );
void function2( );

#pragma startup function1
#pragma exit function2

int main( )
{
   printf ( "\n Now we are in main function" );
   return 0;
}

void function1( )
{
   printf("\nFunction1 is called before main function call");
}

void function2( )
{
   printf ( "\nFunction2 is called just before end of main function" );
}
```

Output

Function1 is called before main function call
Now we are in main function
Function2 is called just before end of main function

PREPROCESSOR OPERATORS

- **The C preprocessor offers following operators to help you in creating macros:**

- **Macro Continuation (\)**
  - A macro usually must be contained on a single line.
  - The macro continuation operator is used to continue a macro that is too long for a single line. For example:

```
#define  message_for(a, b)  \
    printf(#a " and " #b ": We love you!\n")
```

PREPROCESSOR OPERATORS

Stringize (#)

- The stringize or number-sign operator ('#'), when used within a macro definition, converts a macro parameter into a string constant.
- This operator may be used only in a macro that has a specified argument or parameter list. For example:

```
#include <stdio.h>

#define  message_for(a, b)  \
    printf(#a " and " #b ": We love you!\n")

int main(void)
{
   message_for(Carole, Debra);
   return 0;
}
```

- When the above code is compiled and executed, it produces the following result:

```
Carole and Debra: We love you!
```

PREPROCESSOR OPERATORS

Token Pasting (##)

- The token-pasting operator (##) within a macro definition combines two arguments. It permits two separate tokens in the macro definition to be joined into a single token. For example:

```
#include <stdio.h>

#define tokenpaster(n) printf ("token" #n " = %d", token##n)

int main(void)
{
   int token34 = 40;

   tokenpaster(34);
   return 0;
}
```

- When the above code is compiled and executed, it produces the following result: token34 = 40
- How it happened, because this example results in the following actual output from the preprocessor:

```
printf ("token34 = %d", token34);
```

- This example shows the concatenation of token##n into token34 and here we have used both **stringize** and **token-pasting**.

PREPROCESSOR OPERATORS

The defined() Operator

- The preprocessor **defined** operator is used in constant expressions to determine if an identifier is defined using #define. If the specified identifier is defined, the value is true (non-zero). If the symbol is not defined, the value is false (zero). The defined operator is specified as follows:

```
#include <stdio.h>

#if !defined (MESSAGE)
   #define MESSAGE "You wish!"
#endif

int main(void)
{
   printf("Here is the message: %s\n", MESSAGE);
   return 0;
}
```

- When the above code is compiled and executed, it produces the following result: **Here is the message: You wish!**

PARAMETERIZED MACROS

Parameterized Macros

- One of the powerful functions of the CPP is the ability to simulate functions using parameterized macros. For example, we might have some code to square a number as follows:

```
int square(int x) {
   return x * x;
}
```

- We can rewrite above code using a macro as follows:

```
#define square(x) ((x) * (x))
```

Parameterized Macros

- Macros with arguments must be defined using the **#define** directive before they can be used. The argument list is enclosed in parentheses and must immediately follow the macro name. Spaces are not allowed between and macro name and open parenthesis. For example:

```
#include <stdio.h>

#define MAX(x,y) ((x) > (y) ? (x) : (y))

int main(void)
{
   printf("Max between 20 and 10 is %d\n", MAX(10, 20));
   return 0;
}
```

- When the above code is compiled and executed, it produces the following result: **Max between 20 and 10 is 20**