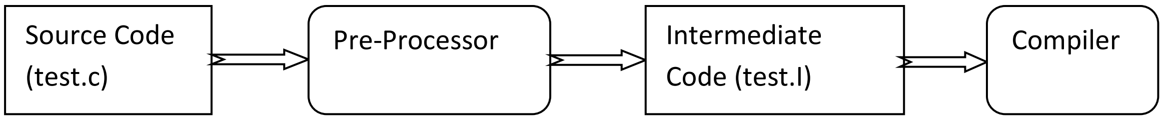
1 What is pre-processor?

- The job of C preprocessor is to process the source code before it is passed to the compiler.



- The pre-processor gets the source code (test.c file) as input and creates expanded intermediate source code (test.I file). This expanded source code is then passed to compiler for compilation.
- The Pre-processor accepts source code as input and is responsible for
  - Removing comments.
  - Interpreting special pre-processor directives denoted by #.

2 List out various pre-processor directives.

- The preprocessor offers several features called preprocessor directives.
- Each of these preprocessor directives begins with a # symbol.
- The directives can be placed anywhere in a program but generally placed at the beginning of a program.
- Following are the preprocessor directives:

Preprocessor Directives	Directive	Description
Macro expansion	#define	Used to define a macro
	#undef	Used to undefine a macro
File inclusion directives	#include	Inserts a particular header from another file
Conditional Directives	#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
Miscellaneous Directives	#error	Prints error message on stderr
	#pragma	Issues special commands to the compiler, using a standardized method

3 What is macro? Explain #define with example.

- A **macro** is a fragment of code which has been a given a name. Whenever the name is used, it is replaced by the contents of the macro.
- You may define any valid identifier as a macro, even if it is a 'C' keyword the pre-processor does not know anything about keyword.
- In Macro Substitution an identifier in a program is replaced by a pre defined string composed of one or more tokens. We can use the #define directive for this purpose.
- It has the following form:

#define <identifier> <token>

- The definition should start with the keyword **#define** and should follow by identifier and a token with at least one blank space between them.
- The token may be any text and identifier must be a valid C name. The pre-processor replaces every occurrence of the **identifier** in the source code by **token**.
- There are different forms of macro substitution. The most common form is:
  1) Simple macro substitution.
  2) Argument macro substitution.
- Simple token replacement is commonly used to define constants.

Example:

```
#define  PI   3.1415926
```

Example:

```
#include<stdio.h>
#define TEN 10
void main ()
{
    int a=10;
    if (a == TEN)
    {
            print (“The value of a is 10”);
    }
}
```

Output:

```
    The value of a is 10
```

- Macro function provide faster program running speed of the program fragment. The pre-processor permits us to define more complex and more useful form of replacements it takes the following form(Argument macro substitution).

```
#define   identifier(f1,f2,f3....fn)   token
```

- C source code below shows the appropriate use of macro Function.

Example:

```
#include<stdio.h>
#include<conio.h>
#define  SQUARE(x)   (x)*(x)
int main()
{
     clrscr();
     printf ("%d", SQUARE(10));
     return 0;
}
```

Output:

```
    100
```

4 What is #undef directive?

- If you have created a macro definition, you can use #undef to remove it. #undef directive causes a defined name to become undefined.
- This means the pre-processor will no longer make anymore text substitutions associated with that word.
- A defined macro can be undefined using following the statement:

  #undef identifier
- For example **#undef VALUE** would cause the definition of **VALUE** to be removed from the system.
- Un-defining macro is useful when we want to restrict the definition only to a particular part of a program. Also to change a definition, you must use **#undef** to undefine it and then use **#define** to reduce it. C Source code below shows the use of **#undef** pre-processor.

Example:

```
#include<stdio.h>
#define  PI   3.14
#define  AREA(x) (PI)*(x)*(x)
int main ()
{
     printf ("%f", AREA(10));
     #undef  PI                    /*PI can no longer be used*/
     return 0;
}
```

Output:

```
     314.000000
```

5 Explain #include directives.

- File inclusion directive causes one file to be included in another. We have already used file inclusion directive before. The pre-processor commands for file inclusion looks like this:

  #include "File name"

  OR

  #include <file name>
- If file name is enclosed within angle brackets, the file is searched for in the standard compiler include paths.
- If the file name is enclosed within double quotes, the search path is expanded to include the current source directory.
- If in case the file is missing or not found compiler creates error.
- Also it is a general standard that header files are include in angle bracket while user defined files are include in double quotes.

  #include <stdio.h>

  And

  #include "myfile.c"
- The above statement in C will include the file stdio.h and myfile.c in the program. Following C source code shows the use of #include pre-processor:

```
#include   <stdio.h>         /*include a standard header file.*/
#include   "C:myfile.c"       /* Include user defined file.*/
int main ()
{
     printf("Notice the use of #Include pre-processor");
     return 0;
}
```

6 Explain predefined macros used in C language.

- There are five pre-defined ANSI C macros and are always available to the programmer for use as listed in following table.
- They cannot be undefined. Every pre-defined macro name is defined with two underscores as prefix and suffix.
- These macros are useful for finding the system information such as date, time, and file name and line number. A program is illustrated for testing these macros.
- Pre-defined macros in ANSI C:

Pre-Defined Macros	Function
DATE	Displays system date in string format.
TIME	Displays system time in string format.
LINE	Displays line number as an integer.
FILE	Displays current file name in string format.
STDC	In ANSI C the value returned will be non-zero.

- Following example illustrating use of pre-defined ANSI C Macros:

  Example:

```
int main ()
{
    printf ("\nFile name and path:%s", __FILE__ );
    printf ("\nDate and Time created:%s %s", __DATE__ , __TIME__ );
    printf ("\nLine No:%d",__LINE__);
    return 0;
}
```

  Output:

```
    File Name and path: D:PRACTICE\MACRO.C
    Date and Time file created: Jan 26 2013 20:28:01
    Line No: 6
```

7 List out Standard I/O Pre-defined Streams in stdio.h.

- There are numerous library functions available for I/O within the stdio.h file. These can be classified into three broad categories:

Console I/O functions	Functions to receive input from keyword and write output to VDU.
File I/O functions	Functions to perform I/O operations on floppy disk or hard disk.

Console I/O functions

Formatted
scanf()
printf()

Unformatted
getch()
putch()
getche()
getchar()
putchar()
gets()
puts()

8 List out pre-defined macros in ctype.h.

- The header file 'ctype.h' also contains a set of macros. Following table lists all the macros available in ctype.h header file. These macros take an argument of integer type and return an integer.

Macro	**Returns True (!0) Value If**
isalpha(d)	d is a letter
isupper(d)	d is a capital letter
islower(d)	d is a small letter
isdigit(d)	d is a digit

isalnum(d)	d is a letter or digit
isxdigit(d)	d is a hexadecimal digit
isspace(d)	d is a space
ispunct(d)	d is a punctuation symbol
isprint(d)	d is a printable character
isgraph(d)	d is a printable, but not be a space
iscntrl(d)	d is a control character
isascii(d)	d is an ASCII code

9 Compare Macro and Function.

Macro	Function
Macros are just a text substitution tool.	In C when one function makes call to another function it is done in several steps which takes time such as saving of system space, stack loading etc.
This increases size of your program.	This decreases size of your program.
Best efficiency and result is achieved from a macro when they are short and frequently used.	Best efficiency and result is achieved from a function when they are long and complex.
Macros are not executable.	Functions are executable code.