Interesting Facts about Macros and Preprocessors in C

In a C program, all lines that start with # are processed by preprocessor which is a special program invoked by the compiler. In a very basic term, preprocessor takes a C program and produces another C program without any #.

Following are some interesting facts about preprocessors in C.
1) When we use ***include*** directive, the contents of included header file (after preprocessing) are copied to the current file.
Angular brackets **<** and **>** instruct the preprocessor to look in the standard folder where all header files are held. Double quotes **"** and **"** instruct the preprocessor to look into the current folder (current directory).
2) When we use ***define*** for a constant, the preprocessor produces a C program where the defined constant is searched and matching tokens are replaced with the given expression. For example in the following program *max* is defined as 100.

```
#include<stdio.h>
#define max 100
int main()
{
    printf("max is %d", max);
    return 0;
}
// Output: max is 100
// Note that the max inside "" is not replaced
```

3) The macros can take function like arguments, the arguments are not checked for data type. For example, the following macro INCREMENT(x) can be used for x of any data type.

```
#include <stdio.h>
#define INCREMENT(x) ++x
int main()
{
    char *ptr = "GeeksQuiz";
    int x = 10;
    printf("%s  ", INCREMENT(ptr));
    printf("%d", INCREMENT(x));
    return 0;
}
// Output: eeksQuiz 11
```

4) The macro arguments are not evaluated before macro expansion. For example consider the following program

```c
#include <stdio.h>
#define MULTIPLY(a, b) a*b
int main()
{
    // The macro is expended as 2 + 3 * 3 + 5, not as 5*8
    printf("%d", MULTIPLY(2+3, 3+5));
    return 0;
}
// Output: 16
```

The previous problem can be solved using following program

```c
#include <stdio.h>
//here, instead of writing a*a we write (a)*(b)
#define MULTIPLY(a, b) (a)*(b)
int main()
{
    // The macro is expended as (2 + 3) * (3 + 5), as 5*8
    printf("%d", MULTIPLY(2+3, 3+5));
    return 0;
}
//This code is contributed by Santanu
// Output: 40
```

5) The tokens passed to macros can be concatenated using operator ## called Token-Pasting operator.

```c
#include <stdio.h>
#define merge(a, b) a##b
int main()
{
    printf("%d ", merge(12, 34));
}
// Output: 1234
```

6) A token passed to macro can be converted to a string literal by using # before it.

```c
#include <stdio.h>
#define get(a) #a
int main()
{
    // GeeksQuiz is changed to "GeeksQuiz"
    printf("%s", get(GeeksQuiz));
}
// Output: GeeksQuiz
```

7) The macros can be written in multiple lines using '\'. The last line doesn't need to have '\'.

```
#include <stdio.h>
#define PRINT(i, limit) while (i < limit) \
                        { \
                            printf("GeeksQuiz "); \
                            i++; \
                        }
int main()
{
    int i = 0;
    PRINT(i, 3);
    return 0;
}
// Output: GeeksQuiz  GeeksQuiz  GeeksQuiz
```

8) The macros with arguments should be avoided as they cause problems sometimes. And Inline functions should be preferred as there is type checking parameter evaluation in inline functions. From C99 onward, inline functions are supported by C language also.
For example consider the following program. From first look the output seems to be 1, but it produces 36 as output.

```
#define square(x) x*x
int main()
{
  int x = 36/square(6); // Expanded as 36/6*6
  printf("%d", x);
  return 0;
}
// Output: 36
```

If we use inline functions, we get the expected output. Also the program given in point 4 above can be corrected using inline functions.

```
inline int square(int x) { return x*x; }
int main()
{
  int x = 36/square(6);
  printf("%d", x);
  return 0;
}
// Output: 1
```

9) Preprocessors also support if-else directives which are typically used for conditional compilation.

```
int main()
{
#if VERBOSE >= 2
  printf("Trace Message");
#endif
}
```

10) A header file may be included more than one time directly or indirectly, this leads to problems of redeclaration of same variables/functions. To avoid this problem, directives like ***defined***, ***ifdef*** and ***ifndef*** are used.

11) There are some standard macros which can be used to print program file (__FILE__), Date of compilation (__DATE__), Time of compilation (__TIME__) and Line Number in C code (__LINE__)

```
#include <stdio.h>

int main()
{
   printf("Current File :%s\n", __FILE__ );
   printf("Current Date :%s\n", __DATE__ );
   printf("Current Time :%s\n", __TIME__ );
   printf("Line Number :%d\n", __LINE__ );
   return 0;
}

/* Output:
Current File :C:\Users\GfG\Downloads\deleteBST.c
Current Date :Feb 15 2014
Current Time :07:04:25
Line Number :8 */
```

12) We can remove already defined macros using :

#undef MACRO_NAME

```
#include <stdio.h>
#define LIMIT 100
int main()
{
   printf("%d",LIMIT);
   //removing defined macro LIMIT
   #undef LIMIT
   //Next line causes error as LIMIT is not defined
   printf("%d",LIMIT);
   return 0;
}
```

Following program is executed correctly as we have declare LIMIT as an integer variable after removing previously defined macro LIMIT

```
#include <stdio.h>
#define LIMIT 1000
int main()
{
   printf("%d",LIMIT);
   //removing defined macro LIMIT
   #undef LIMIT
   //Declare LIMIT as integer again
   int LIMIT=1001;
   printf("\n%d",LIMIT);
   return 0;
}

  /*Output is :
  1000
  1001
  */
```

Another interesting fact about macro using (#undef)

```
#include <stdio.h>
//div function prototype
float div(float, float);
#define div(x, y) x/y

int main()
{
//use of macro div
//Note: %0.2f for taking two decimal value after point
printf("%0.2f",div(10.0,5.0));
//removing defined macro div
#undef div
//function div is called as macro definition is removed
printf("\n%0.2f",div(10.0,5.0));
return 0;
}

//div function definition
float div(float x, float y){
return y/x;
}

/*Output is :
2.00
0.50
*/
```