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 and if the file is not present in current folder, then in standard folder of all header files.

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
```

Run on IDE

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
```

Run on IDE

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

```
#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
```

Run on IDE

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

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

Run on IDE

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

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

Run on IDE

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
```

Run on IDE

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); // Expended as 36/6*6
  printf("%d", x);
  return 0;
}
// Output: 36
```

Run on IDE

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
```

Run on IDE

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

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

Run on IDE

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 */
```

Run on IDE

You may like to take a Quiz on Macros and Preprocessors in C

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : GauravSharma16

Practice Tags : C

Article Tags : C C Basics cpp-macros

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Interesting Facts about Macros and Preprocessors in C
Compiling a C program:- Behind the Scenes
C Programming Language Standard
C Language Introduction
Variables and Keywords in C
Message encryption and decryption using UDP server
Storage of integer and character values in C
How will you print numbers from 1 to 100 without using loop? | Set-2
Common Memory/Pointer Related bug in C Programs
Format specifiers in C

(Login to Rate)

2.4 Average Difficulty : **2.4/5.0**
Based on **223** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Feedback Add Notes

Improve Article

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!