Macros vs Functions

Macros are **pre-processed** which means that all the macros would be processed before your program compiles. However, functions are **not preprocessed but compiled**.

See the following example of Macro:

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

Run on IDE

Output:

```
10
```

See the following example of Function:

```
#include<stdio.h>
int number()
{
    return 10;
}
int main()
{
    printf("%d", number());
    return 0;
}
```

Run on IDE

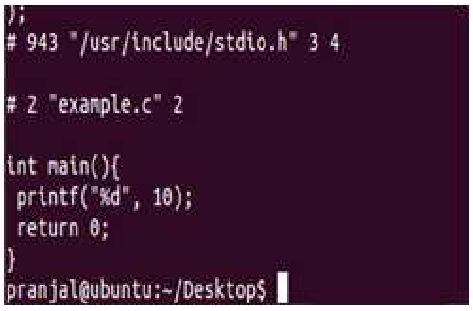
Output:

```
10
```

Now compile them using the command:

```
gcc -E file_name.c
```

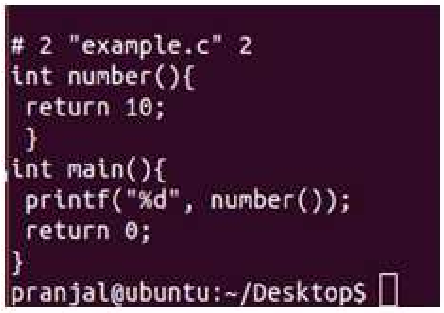
This will give you the executable code as shown in the figure:



```
};
# 943 "/usr/include/stdio.h" 3 4

# 2 "example.c" 2

int main(){
 printf("%d", 10);
 return 0;
}
pranjal@ubuntu:~/Desktop$
```



```
# 2 "example.c" 2
int number(){
 return 10;
 }
int main(){
 printf("%d", number());
 return 0;
}
pranjal@ubuntu:~/Desktop$
```

This shows that the macros are preprocessed while functions are not.

In macros, no type checking(incompatible operand, etc.) is done and thus use of micros can lead to errors/side-effects in some cases. However, this is not the case with functions. Also, macros do not check for compilation error (if any). Consider the following two codes:

Macros:

```
#include<stdio.h>
#define CUBE(b) b*b*b
int main()
{
     printf("%d", CUBE(1+2));
     return 0;
}
```

Run on IDE

Output: Unexpected output

```
7
```

Functions:

```
#include<stdio.h>
int cube(int a)
{
     return a*a*a;
}
int main()
{
    printf("%d", cube(1+2));
    return 0;
}
```

Run on IDE

Output: As expected

```
27
```

- Macros are usually one liner. However, they can consist of more than one line, Click here to see the usage. There are no such constraints in functions.
- The speed at which macros and functions differs. Macros are typically faster than functions as they don't involve actual function call overhead.

Conclusion:
Macros are no longer recommended as they cause following issues. There is a better way in modern compilers that is inline functions and const variable. Below are disadvantages of macros:

a) There is no type checking

b) Difficult to debug as they cause simple replacement.

c) Macro don't have namespace, so a macro in one section of code can affect other section.

d) Macros can cause side effects as shown in above CUBE() example.

See following for more details on macros:
Interesting facts about Macros and Preprocessors

This article is contributed by **Pranjal Mathur.** If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

Recommended Posts: