Preprocessor

Preprocessor is one of the compilation stage, it is used to compile and execute the preprocessor directives.

Features of Preprocessor :

1. Each preprocessor directives starts with '#'.
2. There should be one preprocessor directive for one line.
3. To extend preprocessor into multiple lines, then use ('\') at end.
4. The preprocessor directives doesn't end with semicolon (';').
5. Some of the preprocessor directive are #define,#include,#if,#else,#elif,#endif,#pragma, #ifdef,#if ndef.

Advantages :

1. Preprocessor enhances the code readability. (better code readability)
2. Program modification becomes simple.
3. Easy to debug.
4. Easy to test the program.

Operations of Preprocessor :

Following operations are performed by preprocessors are,

1. Includes the header files.
2. Removes the comments.
3. Replacement of macros.
4. Performs conditional compilation.

1. Includes the header files : Includes the header files is in 2 ways

#include< >

Here, the pre-prossessor searches for header files in a predefined path /usr/include/

#include " "

Here, the pre-prossessor searches the header file in a present working directory, if it is not found then it searches predefined path /usr/include/

```
#include<stdio.h>
//#include<fun.c>
#include"fun.c"
int main()
{
        printf("In main()...\n");
        fun();
}
```

Prog.c

```
#include<stdio.h>
void fun()
{
        printf("fun function\n");
}
```

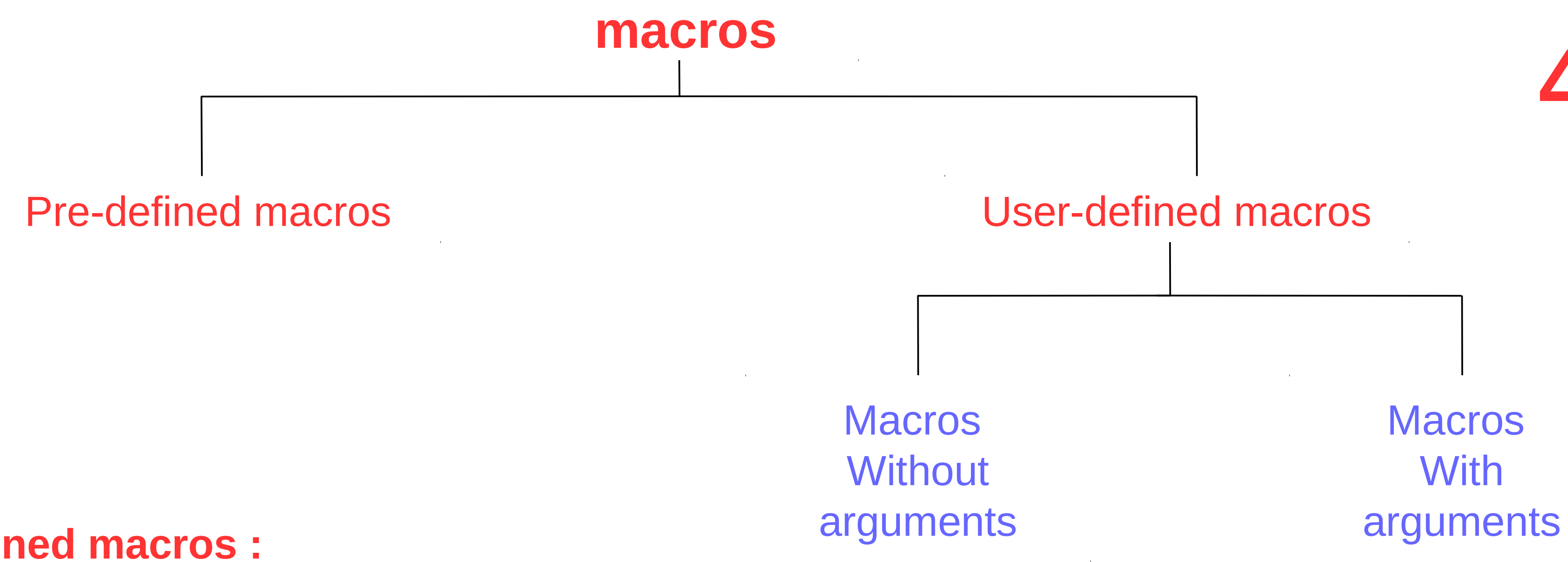
fun.c

Removing the comments :

1. comments are removed in preprocessor stage.
2. They are defined for better code readabilty.

Replacement of macros :

1.It is a part of the program,macro name is replaced with macro body.
2.#define macro_name macro_body.
3.Here, #define is preprocessor directive.



Pre-defined macros :

```
#include<stdio.h>
int main()
{
        printf("%d\n",(int)NULL);
        printf("%d\n",EOF);
        printf("line no : %d\n",__LINE__);
        printf("file name : %s\n",__FILE__);
        printf("time : %s\n",__TIME__); //last compilation time
        printf("date : %s\n",__DATE__); //last compilation date
}
```

User defind macros without arguments :

```
#include<stdio.h>
#define pi 3.14
int main()
{
        printf("%f\n",pi);
}
```

```
#include<stdio.h>
#define char int
#define int float
int main()
{
        char ch; //int ch;  //float ch;
        int x;   //float x;
}
```

Output :

```
float ch;
float x;
```

```c
#include<stdio.h>
#define char int
#define int float
#define float char
int main()
{
        char ch; //int ch; //float ch; //char ch;
        int x; //float x; //char x; //int x;
        float f; //char f; //int f; //float f;
}
```

Output :

```
char ch;
int x;
float f;
```

User defind macros with arguments :

```c
#include<stdio.h>
#define sum(a,b) a+b
int main()
{
        int x = 10,y = 20;
        printf("x+y = %d\n",sum(x,y));
}
```

Output :

```
x+y = 30
```

Macro vs Functions

Macro :

1. Macros are replacement in preprocessor stage.
2. Macro arguments are generic.
3. Stack memory not used in macros.
4. faster execution.

Function :

1. Functions are not for replacements.
2. Function arguments are specific.
3. When a function is called internally stack memory is used for storing the return addresses.
4. saving memory.

Macro vs Typedef

Macro

Syntax #define macro macro_body

1. #define is a preprocess directive
2. Macroname and Macrobody possible to take as keywords.
3. Macros for replacement.

Typedef

Syntax typedef existing_type new_type

1. typedef is a keyword.
2. existing_type is possible as keyword but new_type can not be a keyword.
3. typedef is not for replacement.

```c
#include<stdio.h>
int main()
{
        //typedef int* INT_PTR;
        #define INT_PTR int*

        int x = 10,y = 20;

        INT_PTR p,q; //int *p,q;
        p = &x;
        q = &y;

        printf("*p = %d  *q = %d\n",*p,*q);
}
```

```c
#include<stdio.h>
#define swap(a,b) a=a+b-(b=a)
int main()
{
        char ch1 = 'A',ch2 = 'B';
        int x = 10,y = 20;
        swap(ch1,ch2);
        swap(x,y);
        printf("ch1 = %c  ch2 = %c\n",ch1,ch2);
        printf("x = %d  y = %d\n",x,y);
}
```

```
#include<stdio.h>
#define mul(a,b) a*b
/*
int mul(int a,int b)
{          //5    -1
        return a*b;  //return 5*-1
}
*/
int main()
{
        int x = 2, y = 3;
        printf("%d\n",mul(x+y,x-y));
}
//x+y*x-y ---> 2+3*2-3 ---> 2+6-3 ---> 8-3 ---> 5
```

```
#include<stdio.h>
int main()
{
        printf("vector""india""pvt""ltd""\n");
        printf("vector""""india""""pvt""""ltd""""\n");
        //printf("vector"""india""""pvt""""ltd""""\n");
}
```

```
//stringizing operator (#)
#include<stdio.h>
#define m(x) #x
int main()
{
        m(hello); //--> "hello"
        m(1234);  //--> "1234"
        m(22.7);  //--> "22.7"
}
```

```
#include<stdio.h>
#define print(a) printf(#a" = %d\n",a)
int main()
{
        int x = 10, y = 20, z = 30;
        print(x);
        print(y);
        print(z);
}
/*output :  print(a) --> printf("x = %d\n",x);
  x = 10                printf(#a" = %d\n",a);
  y = 20                printf("x"" = %d\n",x);
  z = 30
*/
```

```
#include<stdio.h>
#define print(a,b,c) printf(#a" = %d\n"#b" = %d\n"#c" = %d\n",a,b,c)
int main()
{
        int x = 10, y = 20, z = 30;
        print(x,y,z);
}
//print(x,y,z)--> printf("x = %d y = %d z = %d\n",x,y,z);
//print(a,b,c)-->
//printf(#a" = %d\n"#b" = %d\n"#z" = %d\n",a,b,c);
//printf("x"" = %d\n""y"" = %d\n""z"" = %d\n",x,y,z);
```

```
//token pasting operator(##)
#include<stdio.h>
#define paste(x,y) x##y
#define marks(x) marks_##x
int main()
{
        int xy = 100;
        float marks_c = 92.5, marks_unix = 73.9;
        printf("xy = %d\n",paste(x,y));
        printf("marks_c = %f\n",marks(c));
        printf("marks_unix = %f\n",marks(unix));
}
```

Conditional compilation

1. It is possible to compile a program based on condition.
2. Preprocessor directives used for conditional compilation are,
   #ifdef , #ifndef , #if , #else , #elif , #endif.

Advantages:
Code size will be minimized.
Disadvantages:
we cannot provide the values at runtime,everything will be decided at preprocessor stage.

```
#include<stdio.h>
#define v1 5
#define v2 2
int main()
{
        #if v1>v2
        printf("v is greater\n");
        #elif v1<v2
        printf("v is smaller\n");
         #endif
}
```

```
#include<stdio.h>
int main()
{
        #define FLAG 8
        #ifdef FLAG
        printf("FLAG is defined\n");
        #endif

         #undef FLAG
         #ifndef FLAG
         printf("FLAG is undefined\n");
         #endif

}
```