Preprocessor - Directives: (#)

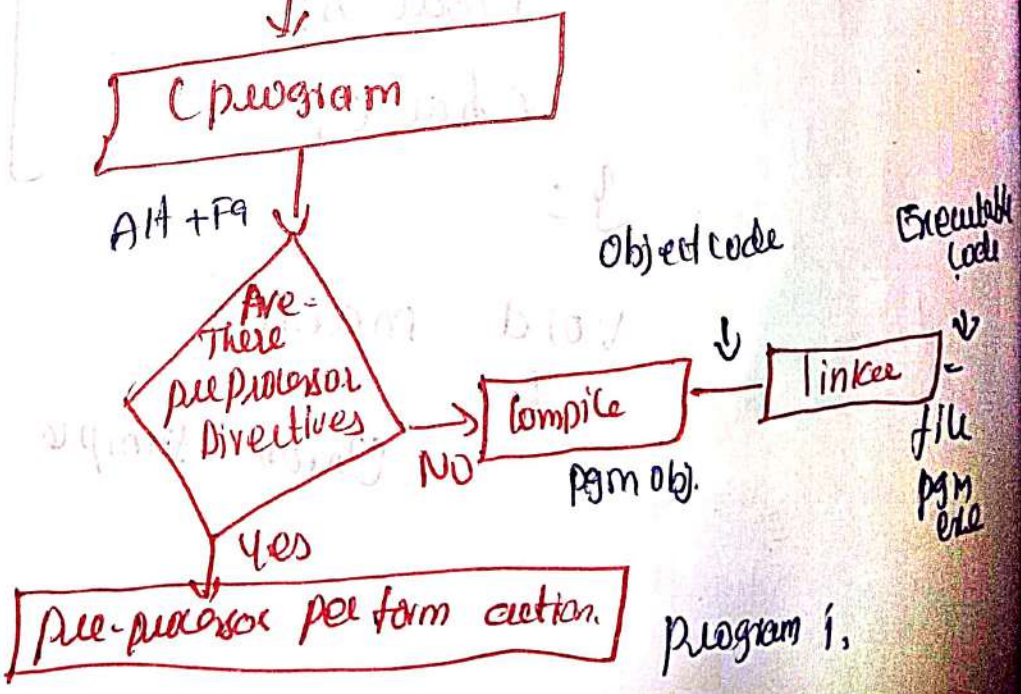
↳ It is a program.
→ not a part of Compiler.
→ It is a step of compiler process
support a

→ In simple terms, a C preprocessor is just a text substitute. tool and it instructs the compiler to do required pre processing before actual compilation.

All the pre processor commands begin with a hash symbol (#).

Diagram → Preprocessor in C

C source code (program.c)



Program 1.

There are 4- main types.

1. Macros.
2. File inclusion
3. Conditional compilation.
4. Other directives.

Macros:

→ The "#define" directives is used to define a macro.

→ Macros are ~~pieces~~ pieces of code in a program that is given same name.

→ When ever this name is encountered by the compiler, the compiler replaces the name with the value.

Syntax: #define macro_name value

eg: #define a 10.

Program:

```
#include<stdio.h>
#define a 10
void main()
{
  printf("%d", a);
}
```

O/p: 10

#undef:

```
#include<stdio.h>
#define a 10
#undef a
void main()
{
  printf("%d", a);
}
```

O/p: error msg.

(2)

File Inclusion:

→ This type of preprocessor directives tells the Compiler to include a file in the source code program.

→ There are two types of files that can be included by the user in the program.

→ Header files or Standard files. These files contain definitions of pre-defined functions like printf(), scanf(), etc.

→ These files must be included to work with these fun.

→ Diff. fun. are declared in diff. header files for-eg: Std I/O fun. are in the "stdio.h" file ~~whereas~~ whereas functions that perform string operations are in the "string.h" files:

Syntax:

#include < file_name >

or

#include " file_name ".

eg. #include <stdio.h>

Programe:

```
#include <stdio.h>
Void main ()
{
printf("Hello world !");
}
```

O/P: Hello world !.

③ Conditional compilation :-

It is a types of directive that helps to compile a specific portion of the program or to skip the compilation of some specific part of the program based on some conditions.

⇒ This can be done with the help of the preprocessing commands.

Six types:

1. # if def
2. # if ndef
3. # if
4. # else
5. # el if
6. # end if

① # if def : The #ifdef preprocessor directives checks if macro is defined by # define. If yes, only then it executes the code, otherwise # else code is executed.

Syntax:

```
# if def macro
# code
# endif
```

ifdef.

```
#include <stdio.h>
# define a 10
void main()
{
#ifdef a
    printf("hello guys");
# endif
}
```

o/p:

Hello guys

#ifndef:

```
#include <stdio.h>
#define a 10
Void main()
{
#ifndef a
  printf("hello");
#endif
}
```

#if → Check for the specified condition.

#else → Alternat code that executes when #if fails.

#endif → used to break the end of #if, #ifdefe #ifndef.

Sample program for: #if, #elif #else:

```
#include <Stdio.h>
#define a 15
Void main()
{
#if a<10
  printf("Value is lessthan 10");
#elif 20
  printf("Value is lessthan 20");
#else
  printf("number not found");
#endif
}
```

Other directives:

1. Undef. Directive.
2. Pragma Directive.

Undef Directives.

↳ It is used to undefine a macro.

program

```
#include <stdio.h>
#define a 10
#undef a
{ printf("%d", a);
};
```

o/p: error message.

Pragma Directions:

It is a special purpose directive and is used to turn on or off some ~~functions~~ features. This type of directives are compiler-Specific.

Some of the # pragma directives are discussed below:

pragma startup.(or) message: ↳ used at compile time to print custom message.

pragma exit (or) once: ↳ to guard the header files. (ie). The header file must be include only once.

pragma warning.
↳ to enable or disable the warning.

Sample program:

```
#include <stdio.h>
#include <conio.h>
Void func ();
 # pragma startup func
 # pragma exit func
Void main()
      {
          printf ("In I am in main");
       getch();
      }
Void func () {
       printf(" I am in func ");
      getch();
      }
```

O/P:

I am in func

I am in main

I am in func.