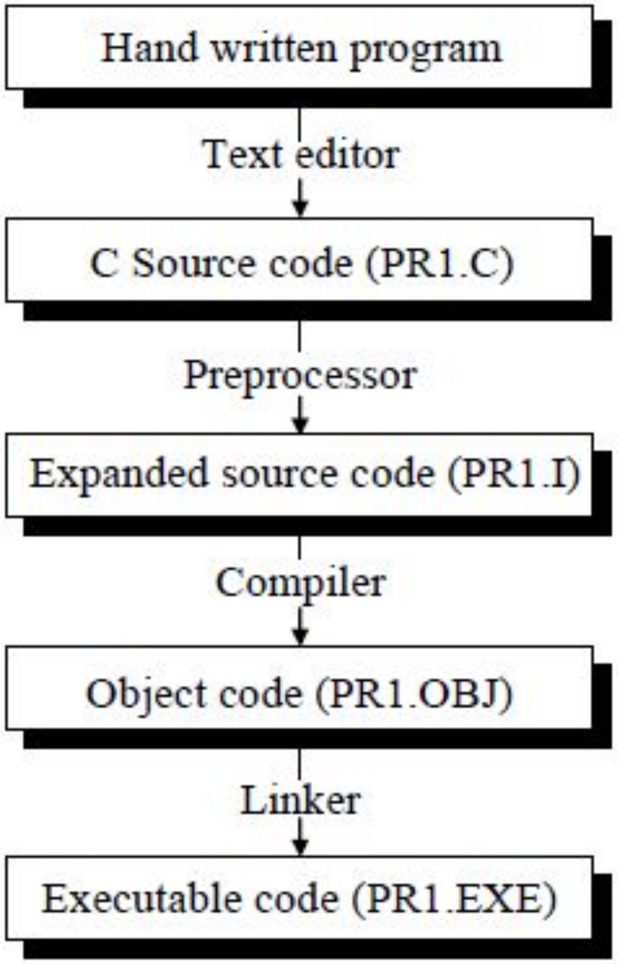
C Preprocessor Directives

Ritu Devi

C Preprocessor statements.

- C preprocessor is a program that processes our source program before it is passed to the compiler
- **Features preprocessor directives**: The preprocessor offers several features called preprocessor directives
  - Each preprocessor directive starts with a # symbol.
  - There can be only one directive on a line.
  - There is no semicolon at the end of a directive.
  - To continue a directive on next line, we should place a backslash at the end of the line.
  - The preprocessor directives can be placed anywhere in a program (inside or outside functions) but they are usually written at the beginning of a program. '
  - A directive is active from the point of its appearance till the end of the program.

- Our program passes through several processors before it is ready to be executed.



- The main functions performed by the preprocessor directives are·
  1. Macros.
  2. File Inclusion.
  3. Conditional Compilation.
  4. Other directives

Macro Expansion: #define

- The general syntax is-

  #define macro_name macro_expansion
- Here macro_name is any valid C identifier, and it is generally taken in capital letters to distinguish it from other variables.
- The macro_expansion can be any text.
- For example-
  - #define PI 3.14159265
  - #define MAX 100
- The C preprocessor searches for macro_name in the C source code and replaces it with the macro_expansion.

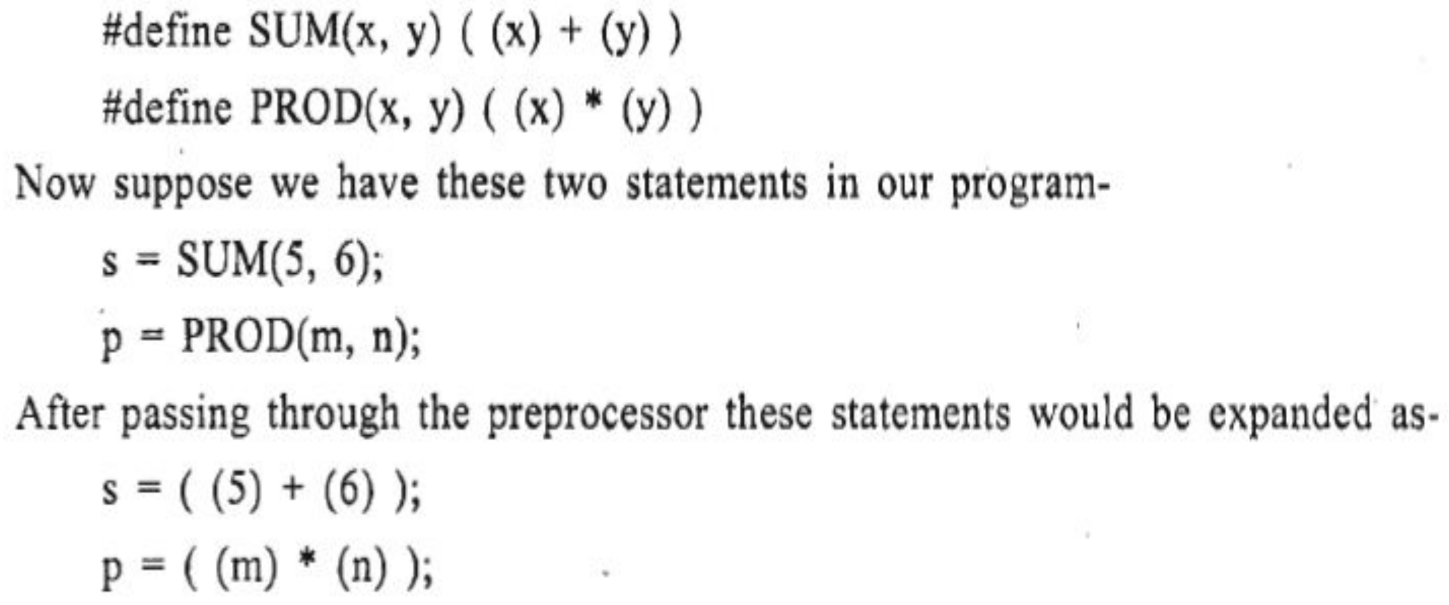
```
#include<stdio.h>
#define MSSG "I understand the use of #define\n"
main( )
{
    printf(MSSG);
}
```

- Some more examples:

```
#define AND &&
#define OR ||
#define BEGIN main( ){
#define END }
#define INFINITE while( 1 )
#define NEW_LINE printf("\n");
#define ERROR printf("An error has occurred\n");
```

Macros with Arguments

- The general syntax is
- #define macro_name(arg 1, arg2, ...) macro_expansion
- For example



```
#define SUM(x, y) ( (x) + (y) )
#define PROD(x, y) ( (x) * (y) )
```

Now suppose we have these two statements in our program-

```
s = SUM(5, 6);
p = PROD(m, n);
```

After passing through the preprocessor these statements would be expanded as-

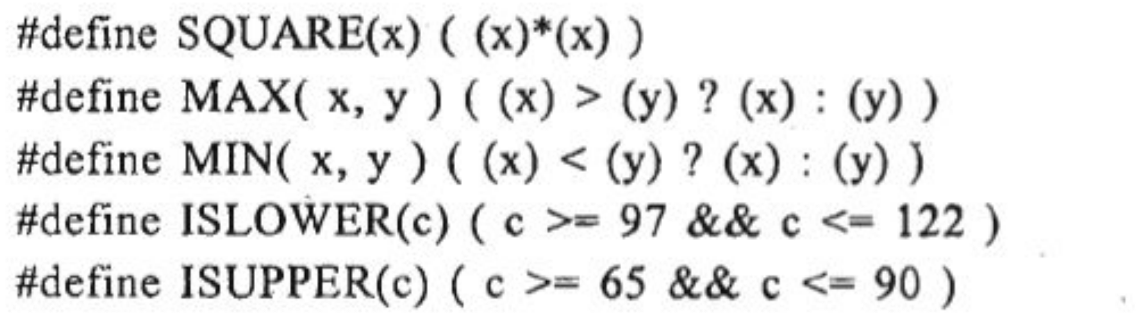
```
s = ( (5) + (6) );
p = ( (m) * (n) );
```

```
#include<stdio.h>
#define  SUM(x,y)  ((x)+(y))
#define  PROD(x,y)  ((x)*(y))
main()
{
    int  l,m,i,j,a=5,b=3;
    float  p,q;
    l=SUM(a,b);
    m=PROD(a,b);
    i=SUM(4,6);
    j=PROD(4,6);
    p=SUM(2.2,3.4);
    q=PROD(2.2,3.4);
    printf("l=%d,m=%d,i=%d,j=%d,p=%.1f,q=%.1f\n",l,m,i,j,p,q);
}
```

Output:

```
l = 8, m = 15, i = 10, j = 24, p = 5.6, q = 7.5
```

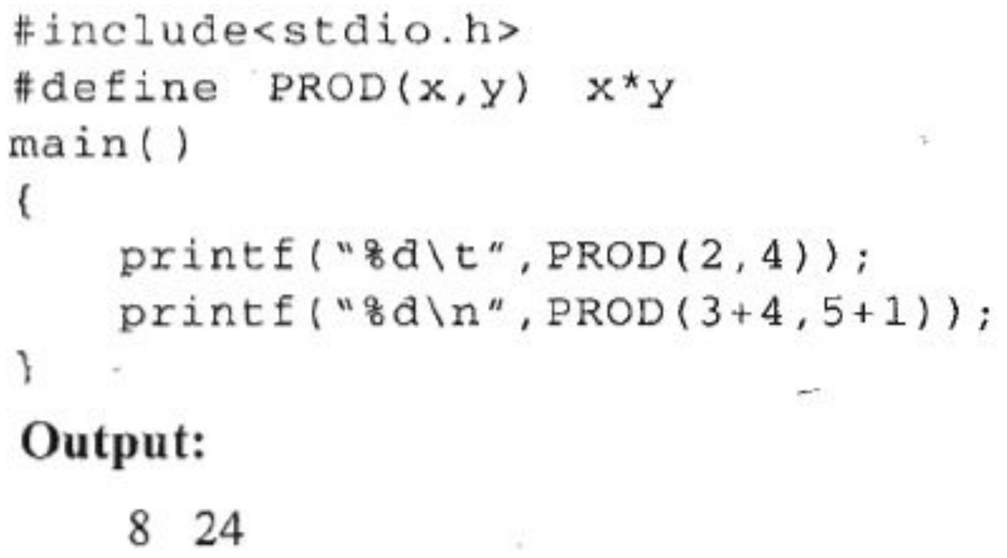
- some more examples:



```
#define SQUARE(x) ( (x)*(x) )
#define MAX( x, y ) ( (x) > (y) ? (x) : (y) )
#define MIN( x, y ) ( (x) < (y) ? (x) : (y) )
#define ISLOWER(c) ( c >= 97 && c <= 122 )
#define ISUPPER(c) ( c >= 65 && c <= 90 )
```

- ***#undef Directive :***
  The definition of a macro will exist from the #define directive tilI the end of the program. If we want to undefine this macro we can use the #undef directive.
  - Syntax: ***#undef macro_name***

Problems with Macros



```c
#include<stdio.h>
#define  PROD(x,y)  x*y
main()
{
    printf("%d\t",PROD(2,4));
    printf("%d\n",PROD(3+4,5+1));
}
```

Output:

```
8  24
```

File Inclusion

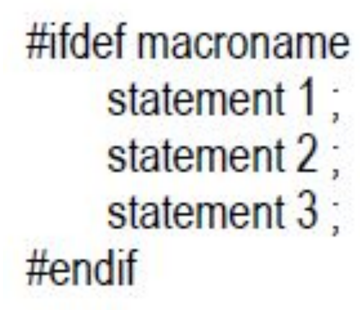
- This type of preprocessor directive tells the compiler to include a file in the source code program.
- The filename should be within angular brackets or double quotes. The syntax is-

```
#include "filename”  /* user defined file*/
#include <filename> /* header file or standard file */
```

- The preprocessor replaces the #include directive by the contents of the specified file

Conditional Compilation

- Conditional Compilation directives are type of directives which helps to compile a specific portion of the program or to skip compilation of some specific part of the program based on some conditions.
- This can be done with the help of two preprocessing commands ‘**ifdef**‘ and ‘**endif**‘.



```
#ifdef macroname
     statement 1 ;
     statement 2 ;
     statement 3 ;
#endif
```

- If the macro with name as ‘*macroname*‘ is defined then the block of statements will execute normally but if it is not defined, the compiler will simply skip this block of statements.

Other directives:

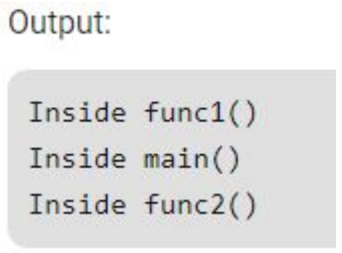
- **#undef Directive**:
- **#pragma Directive**: This directive is a special purpose directive and is used to turn on or off some features.
  - **#pragma startup** and **#pragma exit**: These directives helps us to specify the functions that are needed to run before program startup(before the control passes to main()) and just before program exit (just before the control returns from main()).

```c
#include <stdio.h>

void func1();
void func2();
#pragma startup func1
#pragma exit func2

void func1()
{     printf("Inside func1()\n");
}
void func2()
{
    printf("Inside func2()\n");
}

int main()
{
    void func1();
    void func2();
    printf("Inside main()\n");
      return 0;
}
```



Output:

```
Inside func1()
Inside main()
Inside func2()
```

More problems

1.

```
#define PROD (x,y) ((x)*(y))
main()
{
    int a=3,b=4;
    printf("Product of a and b = %d",PROD(a,b));
}
```

2.

```
#define A 50
#define B A+100
main()
{
    int i,j;
    i=B/20;
    j=500-B;
    printf("i = %d, j = %d\n",i,j);
}
```

3.
```
#define  INFINITE  while(1)
#define  CHECK(a)  if(a==0)  break
main( )
{
    int  x=5;
    INFINITE
    {
        printf("%d  ",x--);
        CHECK(x);
    }
}
```

4.
```
#define  .  ;
main( )
{
    printf("If  the  lift  to  success  is  broken,  ").
    printf("Try  the  stairs.").
}
```

5.
```
#define  CUBE(x)  (x*x*x)
main( )
{
    printf("%d\n",CUBE(1+2));
}
```

6.
```
#define Y 10
main()
{
    #if X || Y && Z
       printf("Sea in Depth\n");
    #else
       printf("See in depth\n");
    #endif
}
```

7.
```
main()
{
    int x=3,y=4,z;
    z=x+y;
    #include<string.h>
    printf("%d\n",z);
}
```

8.
```
#define MAX 3
main()
{
    printf("Value of MAX is %d\n",MAX);
    #undef MAX
    #ifdef MAX
       printf("Have a good day");
    #endif
}
```