Computer Programming & Problem Solving

CS100

Mrs Sanga G. Chaki
Department of Computer Science and Engineering
National Institute of Technology, Goa
May, 2023

Preprocessor Directives

Introduction

1. **Three steps of writing a code:**
   a) **Write**
   b) **Compile**
   c) **execute**
2. **What is a compiler?**
   a) **A special program that converts source code written in human understandable programming languages into machine understandable code, so that execution can take place.**

Preprocessor Directives

1. **What is a preprocessor?**
   a) **Program that processes our source program before it is passed to the compiler**
   b) **Preprocessor commands = Directives**
2. **Types of preprocessor directives:**
   a) **Macro expansion**
   b) **File inclusion**
   c) **Conditional Compilation**
   d) **Miscellaneous directives**

Macro Expansion

```
#define UPPER 25
main( )
{
        int  i ;
        for ( i = 1 ; i <= UPPER ; i++ )
                printf ( "\n%d", i ) ;
}
```

Macro Definition

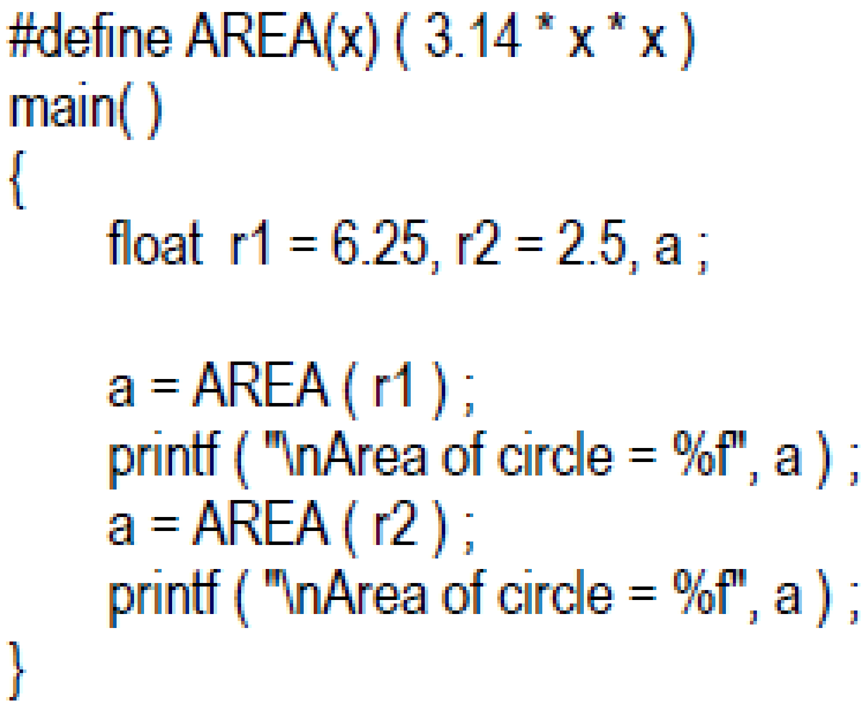
1. **During preprocessing, the preprocessor replaces every occurrence of UPPER in the program with 25.**
2. **UPPER is called the ‘macro templates’, and, 25 is the corresponding ‘macro expansions’**
3. **In C programming it is customary to use capital letters for macro template**

Why use Macros?

1. **Readability increased**
2. **Easy to change values for constants**
3. **Why not use a variable for the same purpose?**
    a) **Macros are efficient for constants – faster**
    b) **A variable may inadvertently get changed in a program**

Macros with Arguments

1. **Macros can have arguments, just as functions**



```
#define AREA(x) ( 3.14 * x * x )
main( )
{
     float  r1 = 6.25, r2 = 2.5, a ;

     a = AREA ( r1 ) ;
     printf ( "\nArea of circle = %f", a ) ;
     a = AREA ( r2 ) ;
     printf ( "\nArea of circle = %f", a ) ;
}
```

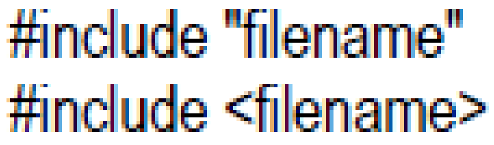
1. **Macro expansion**
2. **Replace x with argument passed**
3. **Calculation done**

Macros vs Functions

1. **Macros take lesser time to execute – literal expansion of macro**
2. **Functions take up lesser space – but arguments passing take time. – proper logical control passing occurs.**
3. **When to use macros? – when it is simple, short, called moderate number of times during a program**
4. **When to use a functions? – larger piece of code, reused fairly often.**

File Inclusion

1. **This directive causes one file to be included in another.**



```
#include "filename"
#include <filename>
```

2. Entire contents of *<u>filename</u>* is inserted into the source code at that point in the program.

3. When is this used?

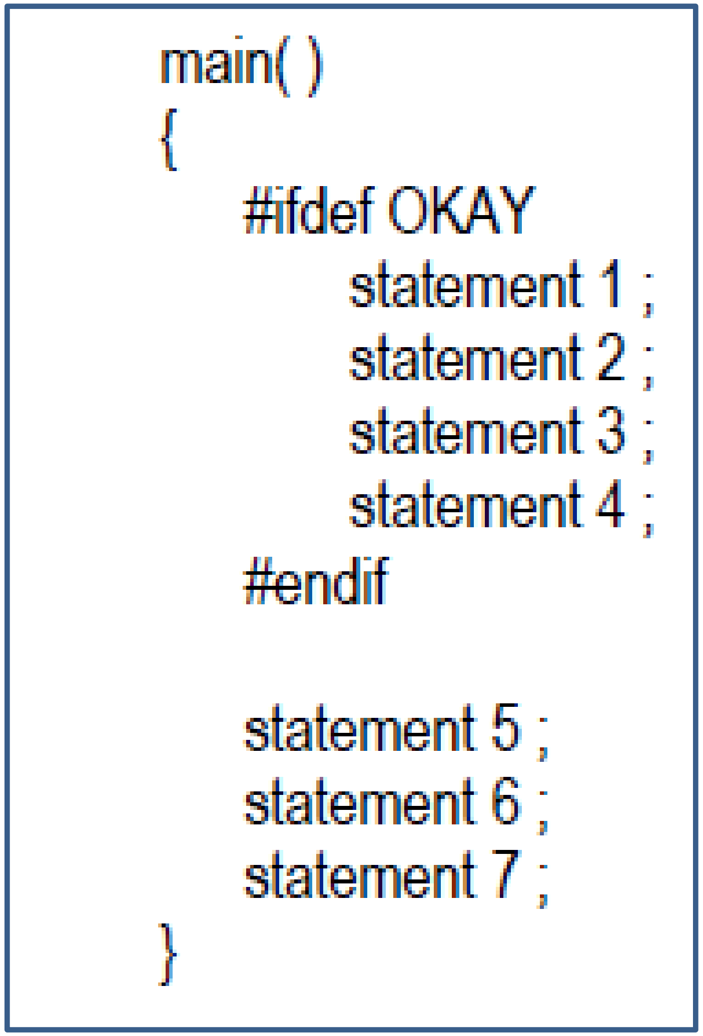
a) To include header files

b) When our source code is too large and we need to divide it into multiple files.

Conditional Compilation

1. **This directive is used when we want the compiler to skip over part of a source code.**
2. **When is this useful?**
   a) **Comment out considerable portions of the code**
   b) **Helps in making code portable**

Conditional Compilation



```
main( )
{
    #ifdef OKAY
        statement 1 ;
        statement 2 ;
        statement 3 ;
        statement 4 ;
    #endif

    statement 5 ;
    statement 6 ;
    statement 7 ;
}
```

1. **Here, statements 1, 2, 3 and 4 would get compiled only if the macro OKAY has been defined.**