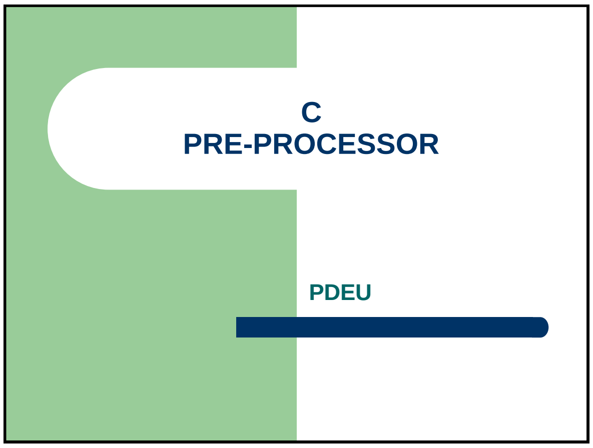
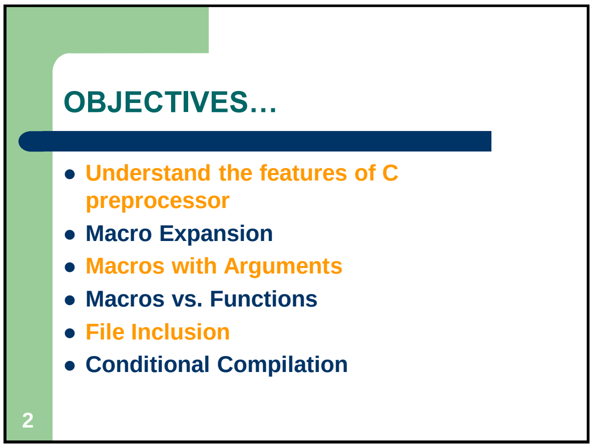


C PRE-PROCESSOR

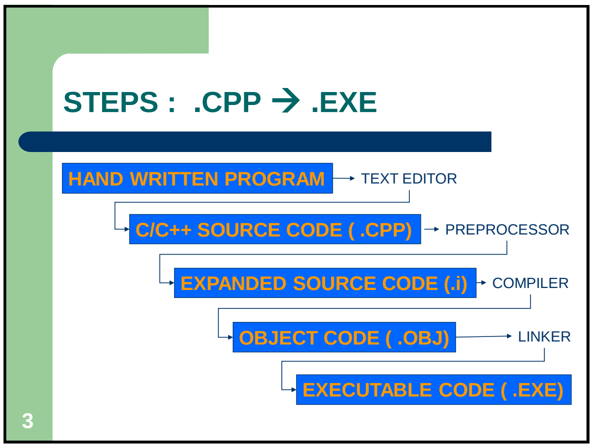
PDEU



OBJECTIVES…

- Understand the features of C preprocessor
- Macro Expansion
- Macros with Arguments
- Macros vs. Functions
- File Inclusion
- Conditional Compilation

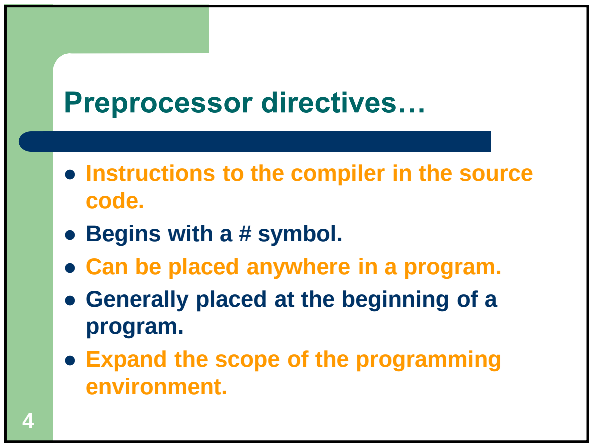
2



STEPS : .CPP → .EXE

$$\begin{tikzcd}
\text{HAND WRITTEN PROGRAM} \arrow[r] & \text{TEXT EDITOR} \arrow[d] \\
\text{C/C++ SOURCE CODE (.CPP)} \arrow[r] & \text{PREPROCESSOR} \arrow[d] \\
\text{EXPANDED SOURCE CODE (.i)} \arrow[r] & \text{COMPILER} \arrow[d] \\
\text{OBJECT CODE (.OBJ)} \arrow[r] & \text{LINKER} \arrow[d] \\
& \text{EXECUTABLE CODE (.EXE)}
\end{tikzcd}$$

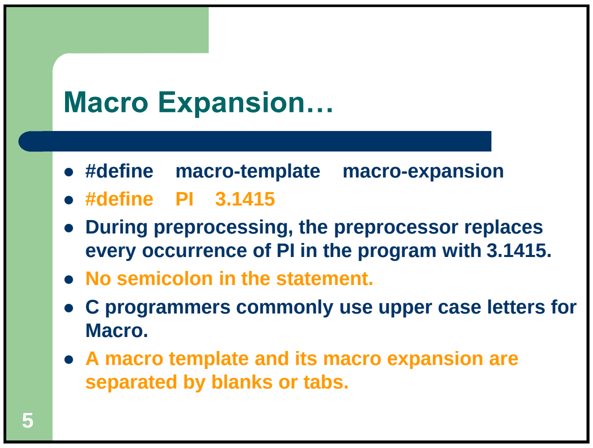
3



Preprocessor directives…

- Instructions to the compiler in the source code.
- Begins with a # symbol.
- Can be placed anywhere in a program.
- Generally placed at the beginning of a program.
- Expand the scope of the programming environment.

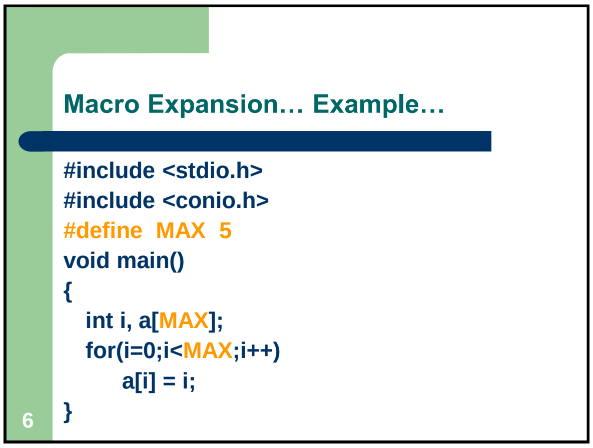
4



Macro Expansion…

- #define macro-template macro-expansion
- #define PI 3.1415
- During preprocessing, the preprocessor replaces every occurrence of PI in the program with 3.1415.
- No semicolon in the statement.
- C programmers commonly use upper case letters for Macro.
- A macro template and its macro expansion are separated by blanks or tabs.

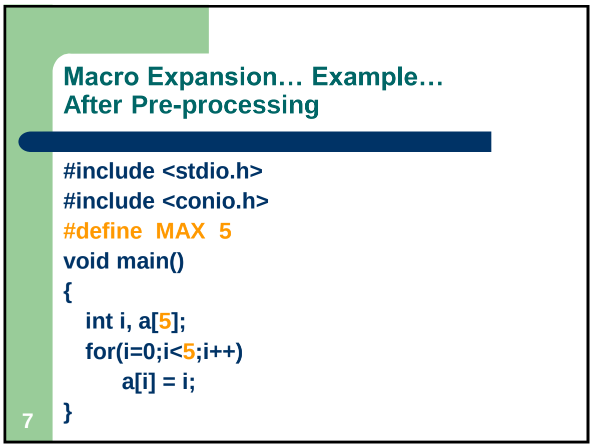
5



Macro Expansion… Example…

```
#include <stdio.h>
#include <conio.h>
#define  MAX  5
void main()
{
   int i, a[MAX];
   for(i=0;i<MAX;i++)
        a[i] = i;
}
```

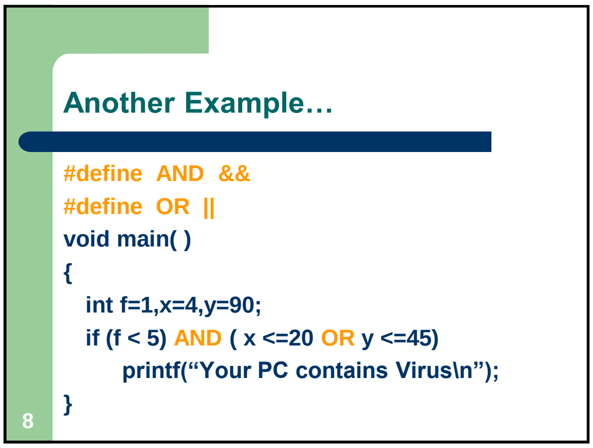
6



Macro Expansion… Example… After Pre-processing

```
#include <stdio.h>
#include <conio.h>
#define  MAX  5
void main()
{
   int i, a[5];
   for(i=0;i<5;i++)
        a[i] = i;
}
```

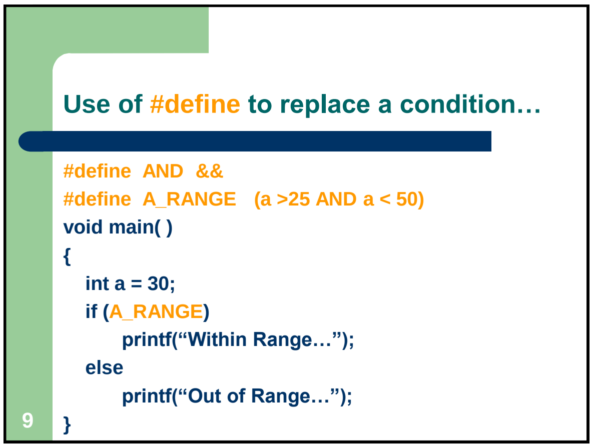
7



Another Example…

```
#define  AND  &&
#define  OR  ||
void main( )
{
   int f=1,x=4,y=90;
   if (f < 5) AND ( x <=20 OR y <=45)
        printf("Your PC contains Virus\n");
}
```

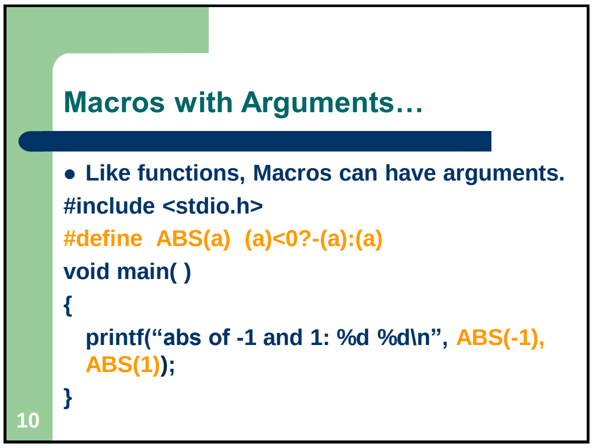
8



Use of #define to replace a condition…

```
#define  AND  &&
#define  A_RANGE   (a >25 AND a < 50)
void main( )
{
   int a = 30;
   if (A_RANGE)
        printf(“Within Range…”);
   else
        printf(“Out of Range…”);
}
```

9

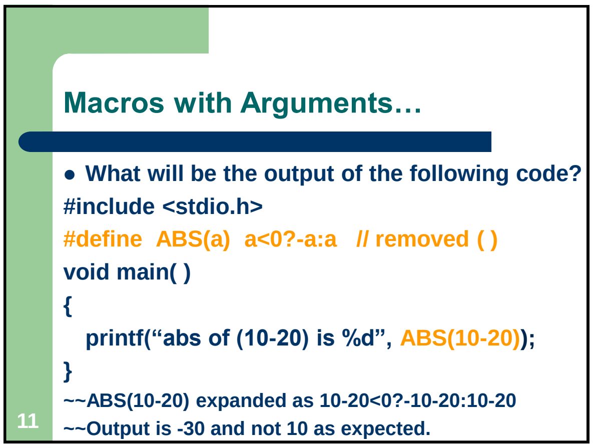


Macros with Arguments…

- Like functions, Macros can have arguments.

```
#include <stdio.h>
#define  ABS(a)  (a)<0?-(a):(a)
void main( )
{
   printf(“abs of -1 and 1: %d %d\n”, ABS(-1),
   ABS(1));
}
```

10



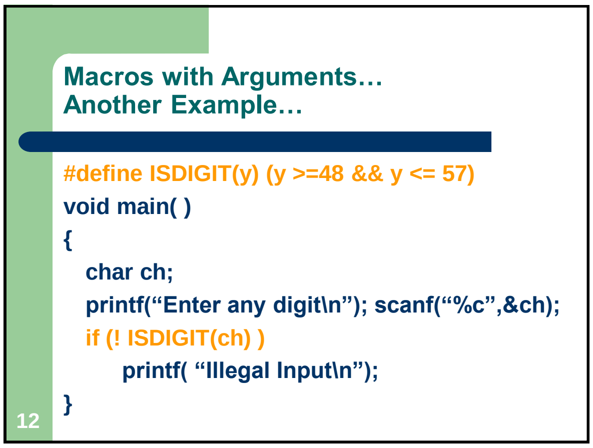
Macros with Arguments…

- What will be the output of the following code?

```
#include <stdio.h>
#define  ABS(a)  a<0?-a:a   // removed ( )
void main( )
{
   printf(“abs of (10-20) is %d”, ABS(10-20));
}
```

~~ABS(10-20) expanded as 10-20<0?-10-20:10-20
~~Output is -30 and not 10 as expected.

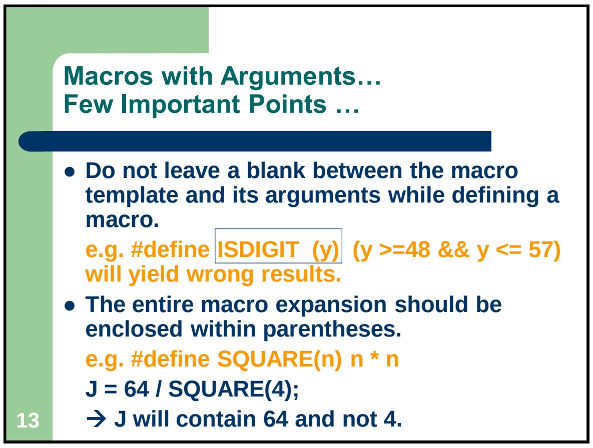
11



Macros with Arguments…
Another Example…

```
#define ISDIGIT(y) (y >=48 && y <= 57)
void main( )
{
   char ch;
   printf(“Enter any digit\n”); scanf(“%c”,&ch);
   if (! ISDIGIT(ch) )
        printf( “Illegal Input\n”);
}
```

12



Macros with Arguments…
Few Important Points …

- Do not leave a blank between the macro template and its arguments while defining a macro.

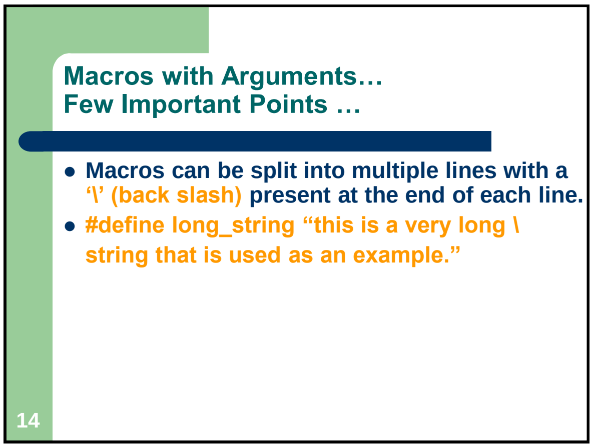
  e.g. #define ISDIGIT (y) (y >=48 && y <= 57) will yield wrong results.
- The entire macro expansion should be enclosed within parentheses.

  e.g. #define SQUARE(n) n * n

  J = 64 / SQUARE(4);

  → J will contain 64 and not 4.

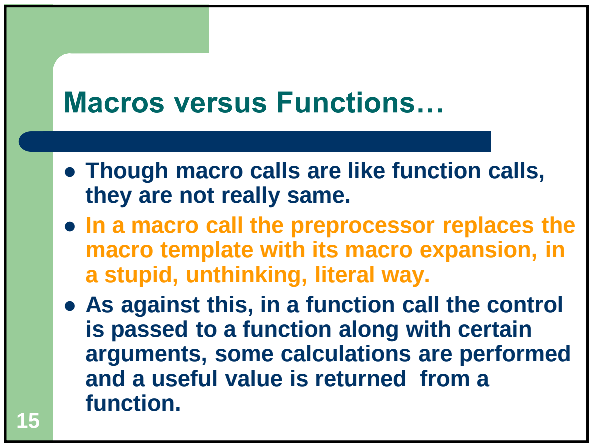
13



Macros with Arguments…
Few Important Points …

- Macros can be split into multiple lines with a '\' (back slash) present at the end of each line.
- #define long_string "this is a very long \
  string that is used as an example."

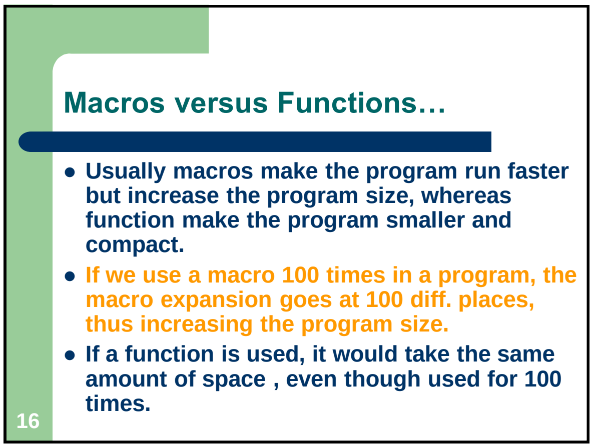
14



Macros versus Functions…

- Though macro calls are like function calls, they are not really same.
- In a macro call the preprocessor replaces the macro template with its macro expansion, in a stupid, unthinking, literal way.
- As against this, in a function call the control is passed to a function along with certain arguments, some calculations are performed and a useful value is returned from a function.

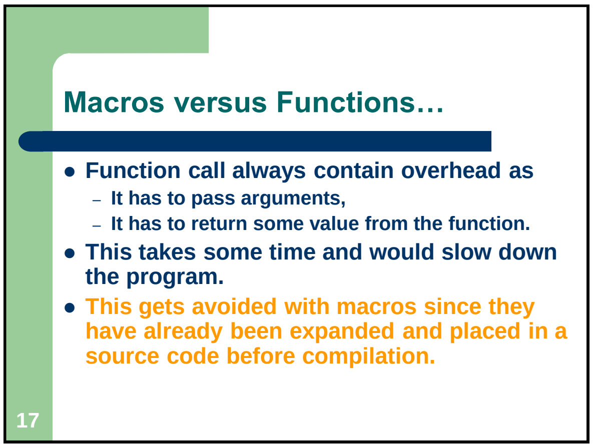
15



Macros versus Functions…

- Usually macros make the program run faster but increase the program size, whereas function make the program smaller and compact.
- If we use a macro 100 times in a program, the macro expansion goes at 100 diff. places, thus increasing the program size.
- If a function is used, it would take the same amount of space , even though used for 100 times.

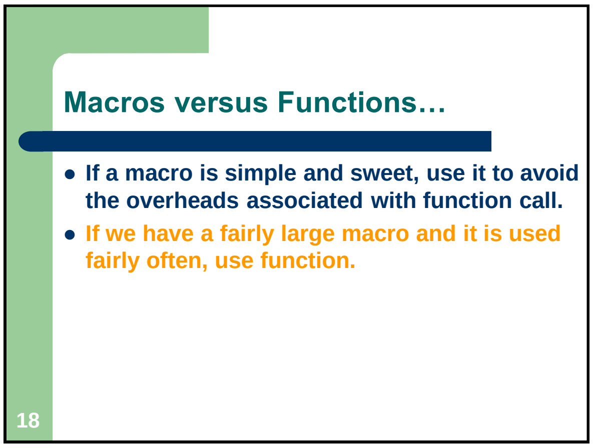
16



Macros versus Functions…

- Function call always contain overhead as
  - It has to pass arguments,
  - It has to return some value from the function.
- This takes some time and would slow down the program.
- This gets avoided with macros since they have already been expanded and placed in a source code before compilation.

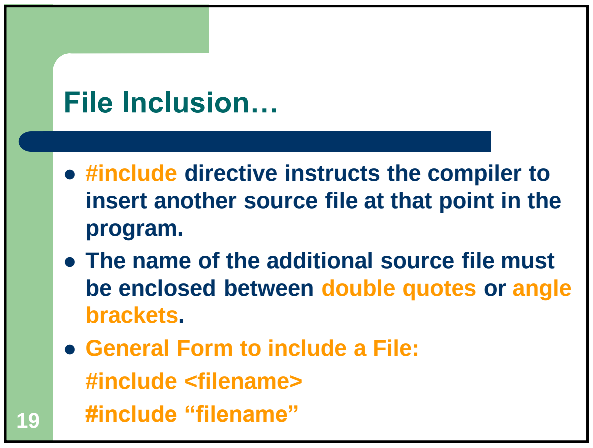
17



Macros versus Functions…

- If a macro is simple and sweet, use it to avoid the overheads associated with function call.
- If we have a fairly large macro and it is used fairly often, use function.

18



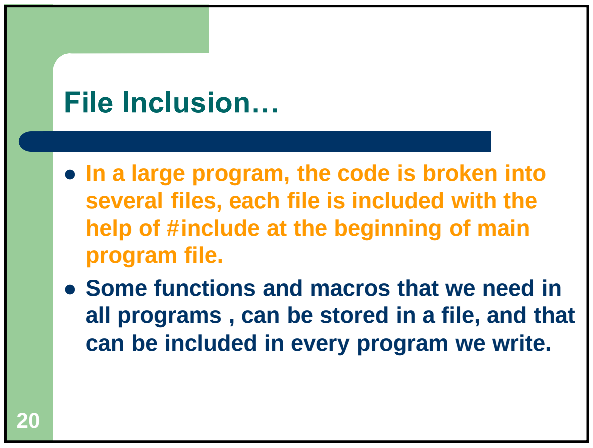
File Inclusion…

- #include directive instructs the compiler to insert another source file at that point in the program.
- The name of the additional source file must be enclosed between double quotes or angle brackets.
- General Form to include a File:

  #include <filename>

  #include “filename”

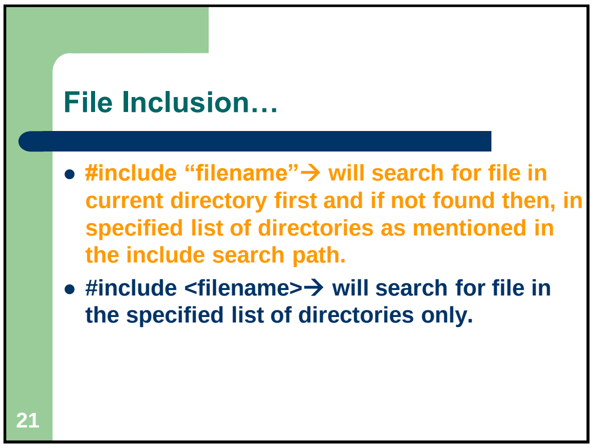
19



File Inclusion…

- In a large program, the code is broken into several files, each file is included with the help of #include at the beginning of main program file.
- Some functions and macros that we need in all programs , can be stored in a file, and that can be included in every program we write.

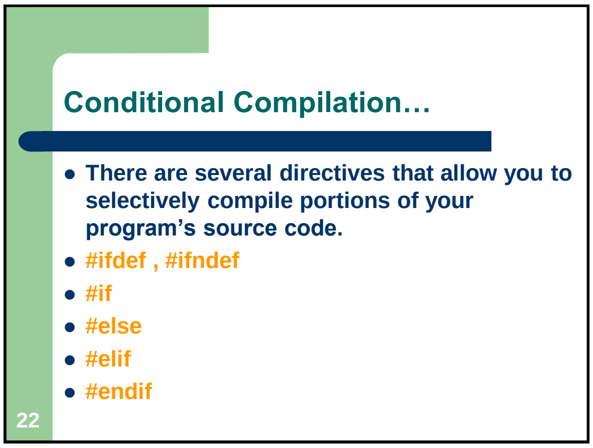
20



File Inclusion…

- #include "filename"→ will search for file in current directory first and if not found then, in specified list of directories as mentioned in the include search path.
- #include <filename>→ will search for file in the specified list of directories only.

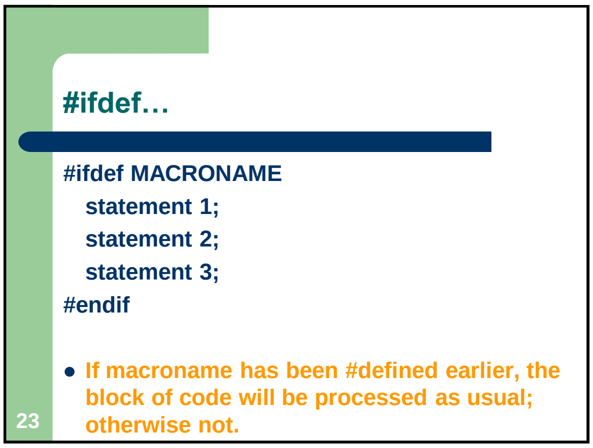
21



Conditional Compilation…

- There are several directives that allow you to selectively compile portions of your program's source code.
- #ifdef , #ifndef
- #if
- #else
- #elif
- #endif

22

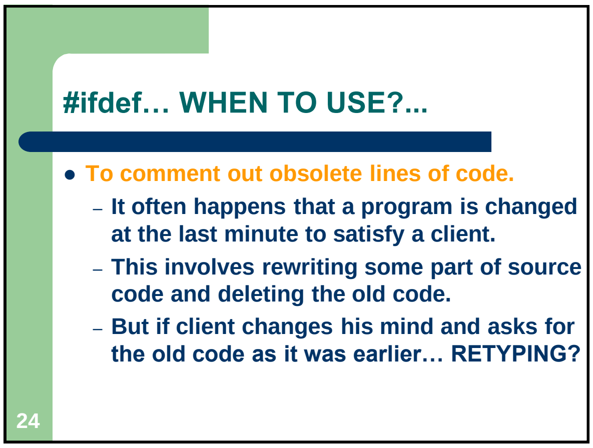


#ifdef…

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

- If macroname has been #defined earlier, the block of code will be processed as usual; otherwise not.

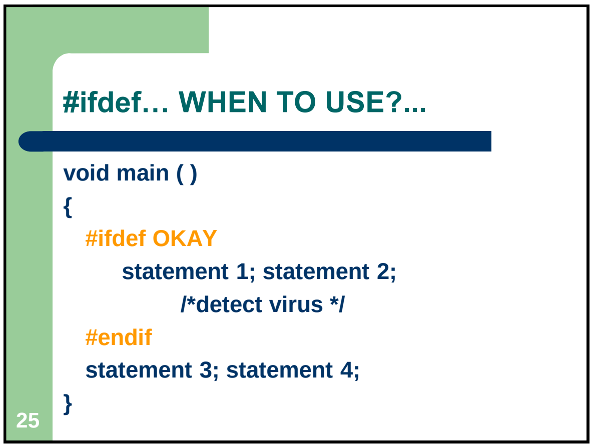
23



#ifdef… WHEN TO USE?...

- To comment out obsolete lines of code.
  - It often happens that a program is changed at the last minute to satisfy a client.
  - This involves rewriting some part of source code and deleting the old code.
  - But if client changes his mind and asks for the old code as it was earlier… RETYPING?

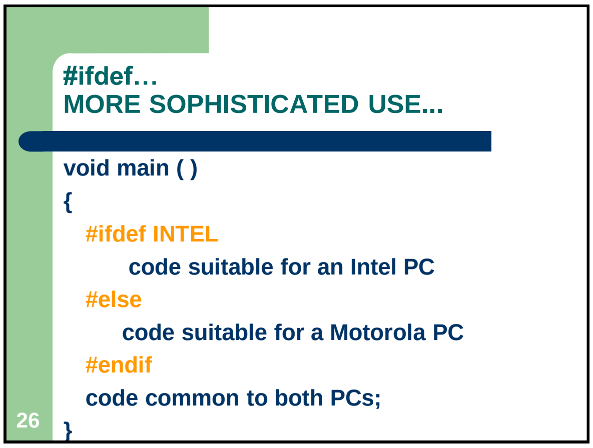
24



#ifdef… WHEN TO USE?...

```
void main ( )
{
   #ifdef OKAY
        statement 1; statement 2;
                /*detect virus */
   #endif
   statement 3; statement 4;
}
```

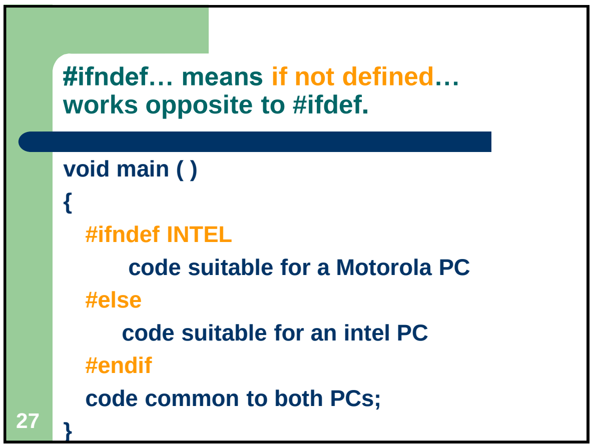
25



#ifdef… MORE SOPHISTICATED USE...

```
void main ( )
{
   #ifdef INTEL
         code suitable for an Intel PC
   #else
        code suitable for a Motorola PC
   #endif
   code common to both PCs;
}
```

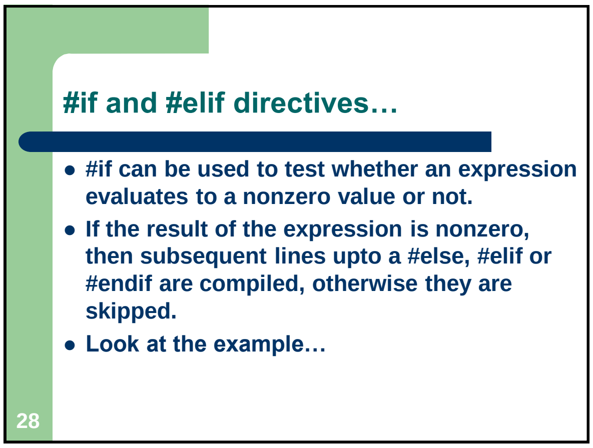
26



#ifndef… means if not defined… works opposite to #ifdef.

```
void main ( )
{
   #ifndef INTEL
         code suitable for a Motorola PC
   #else
        code suitable for an intel PC
   #endif
   code common to both PCs;
}
```

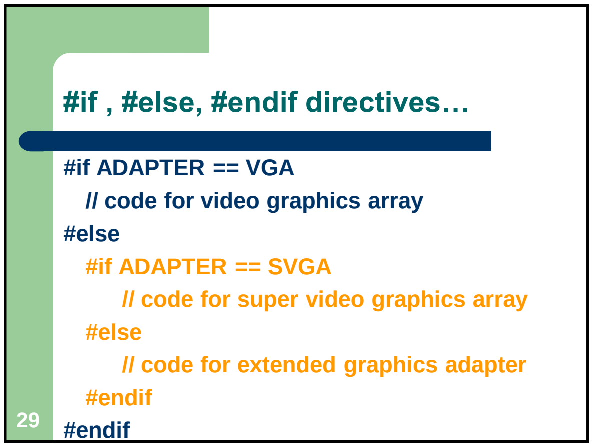
27



#if and #elif directives…

- #if can be used to test whether an expression evaluates to a nonzero value or not.
- If the result of the expression is nonzero, then subsequent lines upto a #else, #elif or #endif are compiled, otherwise they are skipped.
- Look at the example…

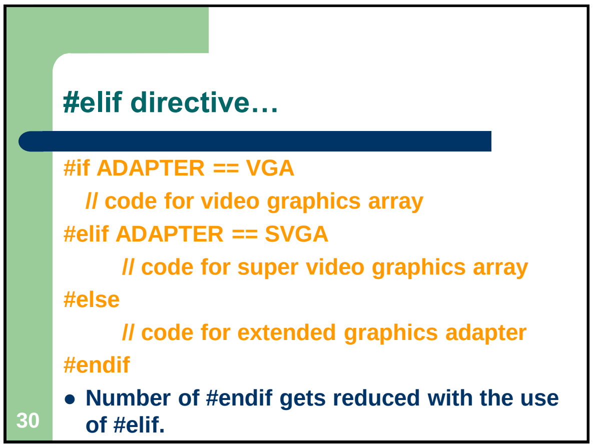
28



#if , #else, #endif directives…

```
#if ADAPTER == VGA
  // code for video graphics array
#else
  #if ADAPTER == SVGA
      // code for super video graphics array
  #else
      // code for extended graphics adapter
  #endif
#endif
```

29

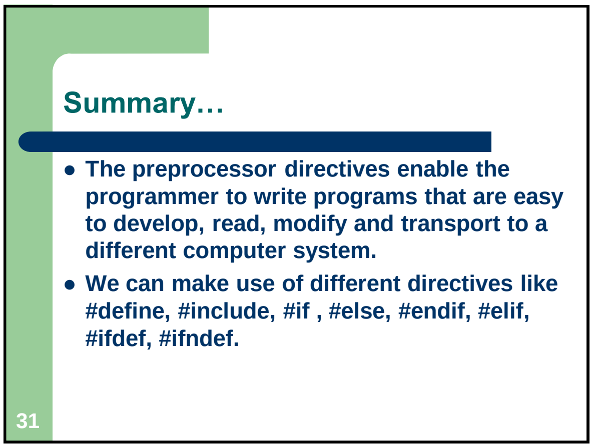


#elif directive…

```
#if ADAPTER == VGA
  // code for video graphics array
#elif ADAPTER == SVGA
      // code for super video graphics array
#else
      // code for extended graphics adapter
#endif
```

- Number of #endif gets reduced with the use of #elif.

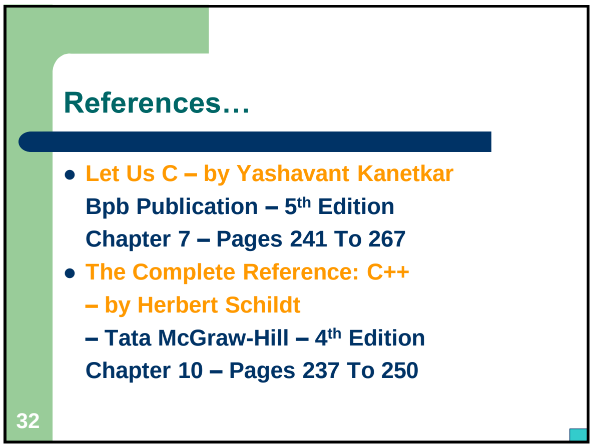
30



Summary…

- The preprocessor directives enable the programmer to write programs that are easy to develop, read, modify and transport to a different computer system.
- We can make use of different directives like #define, #include, #if , #else, #endif, #elif, #ifdef, #ifndef.

31



References…

- Let Us C – by Yashavant Kanetkar
  Bpb Publication – 5th Edition
  Chapter 7 – Pages 241 To 267
- The Complete Reference: C++
  – by Herbert Schildt
  – Tata McGraw-Hill – 4th Edition
  Chapter 10 – Pages 237 To 250

32