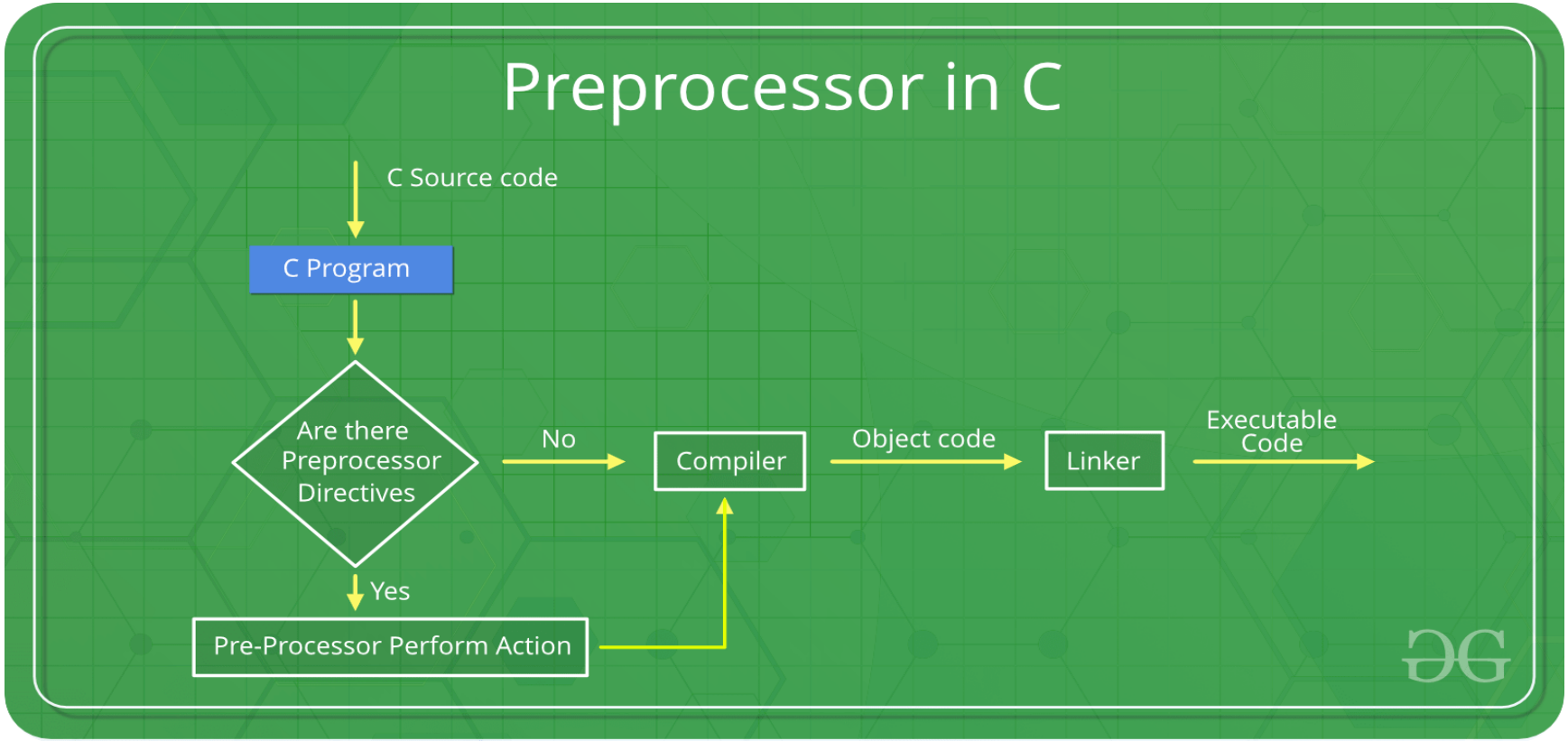
C++ Associate Programing

Preprocessors in C++

Lesson Objectives

- *Macros*
- *File Inclusion*
- *Conditional Compilation*
- *Other Directives*

Intermediate Steps



Section 1

MACROS

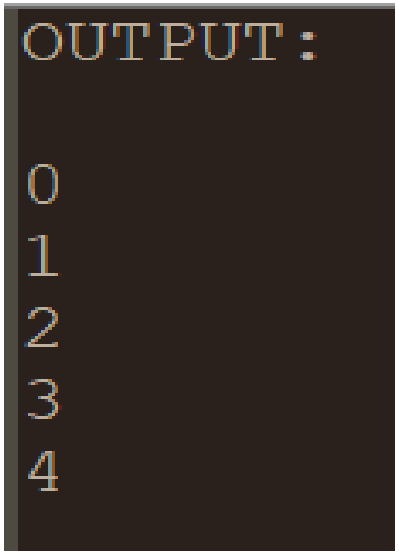
Macros

- *Macros are a piece of code in a program which is given some name.*
- *Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code.*
- *The '#define' directive is used to define a macro.*

Macros: Example

```
#include <iostream>

// macro definition
#define LIMIT 5
int main()
{
    for (int i = 0; i < LIMIT; i++)
    {
        std::cout << i << "\n";
    }
    return 0;
}
```



Macros with arguments

We can also pass arguments to macros. Macros defined with arguments works similarly as functions

```
#include <iostream>
// macro with parameter
#define AREA(l, b) (l * b)
int main()
{
   int l1 = 10, l2 = 5, area;
   area = AREA(l1, l2);
   std::cout << "Area of rectangle is: " << area;
   return 0;
}
```

```
OUTPUT:

Area of rectangle is: 50
```

Section 2

FILE INCLUSION

File Inclusion

This type of preprocessor directive tells the compiler to include a file in the source code program. There are two types of files which can be included by the user in the program:

- *Header File or Standard files*
- *User defined files*

File Inclusion: Header File or Standard files

- *These files contains definition of pre-defined functions like printf(), scanf() etc*
- *These files must be included for working with these functions*
- *Different function are declared in different header files.*

File Inclusion: Header File or Standard files

- ***Syntax:***

```
#include< file_name >
```

- *Where file_name is the name of file to be included.*

- *The ‘<‘ and ‘>’ brackets tells the compiler to look for the file in standard directory.*

File Inclusion: User defined files

- *When a program becomes very large, it is good practice to divide it into smaller files and include whenever needed. These types of files are user defined files.*

- *These files can be included as:*

```
#include"filename"
```

Section 3

CONDITIONAL COMPILATION

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‘.*

Conditional Compilation: Syntax

```
#ifdef macro_name
    statement1;
    statement2;
    statement3;
    .
    .
    .
    statementN;
#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.

Section 4

OTHER DIRECTIVES

Other Directives

Apart from the above directives there are two more directives which are not commonly used. These are:

- *#undef Directive*
- *#pragma Directive*

Other Directives: #undef Directive

- *The #undef directive is used to undefine an existing macro. This directive works as:*

```
#undef LIMIT
```

- *Using this statement will undefine the existing macro LIMIT. After this statement every "#ifdef LIMIT" statement will evaluate to false.*

Other Directives: #pragma Directive

- *In the C and C++ programming languages, #pragma once is a non-standard but widely supported preprocessor directive designed to cause the current source file to be included only once in a single compilation.*
- *And #pragma once serves the same purpose as include guards, but with several advantages, including: less code, avoidance of name clashes, and sometimes improvement in compilation speed.*

Other Directives: #pragma once

Example [edit]

File "grandparent.h"

```
#pragma once

struct foo
{
    int member;
};
```

File "parent.h"

```
#include "grandparent.h"
```

File "child.c"

```
#include "grandparent.h"
#include "parent.h"
```

In this example, the inclusion of grandparent.h in both parent.h and child.c would ordinarily cause a compilation error, because a struct with a given name can only be defined a single time in a given compilation.

The #pragma once directive serves to avoid this by ignoring subsequent inclusions of grandparent.h.

Lesson Summary

- *Macros*
- *File Inclusion*
- *Conditional Compilation*
- *Other Directives*

Thank you

Q&A