C++

session 11

#11

- Preprocessor Directives

Preprocessor Directives

Preprocessor directives are lines included in the code of programs preceded by a hash sign (#). These lines are not program statements but directives for the preprocessor.

Preprocessor Directives

The preprocessor examines the code before actual compilation of code begins and resolves all these directives before any code is actually generated by regular statements.

Preprocessor Directives

Those directives are classed by different categories:

- Macro definitions (#define, #undef)
- Conditional inclusions (#ifdef, #ifndef, #if, #endif, #else and #elif)
- Line control (#line)
- Error directive (#error)
- Source file inclusion (#include)

Preprocessor Directives

Macro Definition

A macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro. There are two kinds of macros. They differ mostly in what they look like when they are used. Object-like macros resemble data objects when used, function-like macros resemble function calls.

Object-like macro
#define **BUFFER_SIZE** 1024
Function-like macro
#define **lang_init()** c_init()

Preprocessor Directives

Conditional inclusions

These directives allow to include or discard part of the code of a program if a certain condition is met.

#ifdef allows a section of a program to be compiled only if the macro that is specified as the parameter has been defined

```
#ifdef TABLE_SIZE
int table[TABLE_SIZE];
#endif
```

Preprocessor Directives

Conditional inclusions

#ifndef serves for the exact opposite: the code between #ifndef and #endif directives is only compiled if the specified identifier has not been previously defined.

```
#ifndef TABLE_SIZE
#define TABLE_SIZE 100
#endif
int table[TABLE_SIZE];
```

Preprocessor Directives

Conditional inclusions

The #if, #else and #elif (i.e., "else if") directives serve to specify some condition to be met in order for the portion of code they surround to be compiled. The condition that follows #if or #elif can only evaluate constant expressions, including macro expressions.

```
#if TABLE_SIZE>200
#undef TABLE_SIZE
#define TABLE_SIZE 200
#elif TABLE_SIZE<50
#undef TABLE_SIZE
#define TABLE_SIZE 50
#else
#undef TABLE_SIZE
#define TABLE_SIZE 100
#endif
int table[TABLE_SIZE];
```

Preprocessor Directives

Line control

When we compile a program and some error happens during the compiling process, the compiler shows an error message with references to the name of the file where the error happened and a line number, so it is easier to find the code generating the error.

```
#line 3 "this is my file"
int a?;
```

```
$> g++ main.cpp
this is my file: In function ‘int main()’:
this is my file:3:10: error: expected initializer before ‘?’ token
```

Preprocessor Directives

Error directive

This directive aborts the compilation process when it is found, generating a compilation error that can be specified as its parameter

```
#ifndef __cplusplus
#error A C++ compiler is required!
#endif
```

Preprocessor Directives

Source file inclusion

When the preprocessor finds an #include directive it replaces it by the entire content of the specified header or file

```
#include <header> //system includes
#include "file" // local includes
```

Preprocessor Directives

Preprocessor directives allows you to change the behavior at compilation time and not during the run-time. This also really useful when you need to deal many systems or architectures with only one source code.

For example, on Linux you know the socket's header is <sys/socket.h> but if you need to support windows in the same time you will be<winsock2.h>.

On your source code if you write

```
#include <sys/socket.h>
#include <winsock2.h>
```

The code will never compile because on of them is missing either or Linux or Windows

Preprocessor Directives

Each compiler provides predefined macros that will only use some files or portions of codes depending on the target you need to compile your binary.

```
#ifdef __linux
#include <sys/socket.h>
#elif __MACH__
#include <sys/socket.h>
#elif __WIN32__
#include <winsock2.h>
#elif __WIN64__
#include <winsock2.h>
#else
#error Cannot detect the target os
#endif
```