International Journal of Scientific Engineering and Research (IJSER)
ISSN (Online): 2347-3878
Index Copernicus Value (2015): 62.86 | Impact Factor (2015): 3.791

New Pre-Processor EXCLUDE Directive of C/C++

Gaurav Kumar Roy

[1]B.S. EC Council University (School of Cyber Security) USA, [2] M.C.A. Lovely Professional University (School of Computer Application) India

Abstract: ***This study and research paper is an attempt to bring in a new feature in C and C++ language where the unused headers can be omitted or excluded from the program. This paper brings in the idea of excluding the unused libraries from within the program. I'd further like to forward my research paper to our respected Sir Bjarne Stroustrup to apply my idea into his compiler.***

Keywords: compiler, libraries, header-file, programming, efficiency, header file, algorithm

1. Introduction

As we all know that today's modern coding techniques and programming methodologies are equipped with more efficiency so that the time and space complexities can be reduced to a handsome extent. Not a single corporation, organization, developer or programmer wants unnecessary codes or function or even libraries to get added up in the program we write. So here I come with one solution to eliminate and reject those unnecessary inclusions, in case we got some within our program. This elimination of unnecessary may get performed based on certain criteria as well. I will discuss in this paper why I thought of this and the algorithm to eliminate and when to implement this concept.

2. Topic of Discussion

Here's a lists of topics to be discussed:

- What is Programming
- Efficiency and Effectiveness of Programming
- The Famous C / C++
- Header files in C / C++
- #include <bits/stdc++.h>
- #exclude: A new Approach to Preprocessors.

3. What is Programming?

Programming is the technique of grabbing an algorithm and encoding it into a specific notation, i.e. a programming language, so that it can be executed by a computer in the form of program. Although many programming languages and many different types of computers exist, the important first step is the need to have the solution.

4. Efficiency and Effectiveness of Programming

A great starting point for talking about program evaluation is to get a better understanding of the concepts of "efficiency", "effectiveness" of a program. In simple terms, program efficiency relates to the cost of producing products or services relative to other programs or to some ideal process. Program effectiveness relates to the level by which the activities of a program produce the desired effect. According to Peter Ferdinand Drucker, who was an Austrian-born American management consultant, educator, and author; and according to him, *Efficiency* is the capacity to do things perfect (which tends to perfection). *Effectiveness* is the capacity to do the right things in a right way/path. These two became the primary important thing in today's programming and tech world as well. Why I'm discussing about these topics will be answered in the coming points and topics.

4.1 Figure to Show its effect

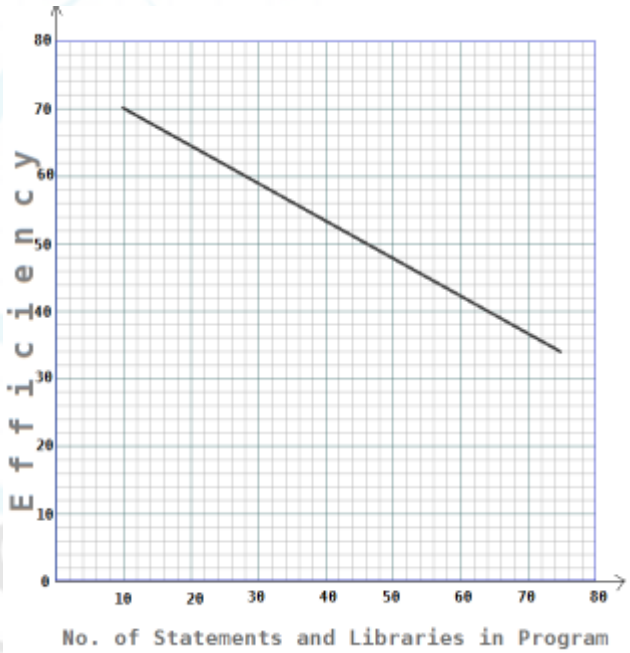

Figure 1: Graph showing the inverse reaction of efficiency wrt increase in Program statements

In the above diagram, it is clear that with the increase in the number of statements or libraries, there lies an opposite outcome, i.e. in the above graph; it decreases the efficiency of a program. So my main focus is to point out how to reduce or minimize such problem by eliminating extra unused statements, expressions and libraries from a program.

5. The Famous C and C++

C is a middle-level programming language that was developed in the mid-1970s. It was originally used for writing UNIX programs, but is now used to write applications for nearly every available platform.

C++, pronounced "C plus plus," is a high-level programming language that was built off the C language. The syntax of C++

is nearly identical to C, but it has object-oriented features, which allow the programmer to create objects within the code. This makes programming easier, more efficient, and some would even say more fun. Because of the power and flexibility of the language, most software programs today are written in C++.

6. Header Files in C and C++

Header files are library files of C and C++ that contain definitions of pre-defined functions and variables that can be imported or used in any C++ program by using the pre-processor directive **#include** statement in the beginning o the program. All header file have an extension ".h" that contains C++ function declaration and macro definition.

Examples of some header files are:

✓ Stdio.h
✓ Iostream.h
✓ Conio.h
✓ Stdlib.h
✓ String.h
✓ Bits/stdc++.h
✓ Algorithm.h
✓ Math.h and lots more….

Each header file contains information (or declarations) for a particular group of functions. Like stdio.h header file contains declarations of standard input and output functions available in C++ which is used for get the input and print the output. Similarly, the header file math.h contains declarations of mathematical functions available in C++.

7. The newly introduced <bits/stdc++.h> Header File

It is basically a header file that includes every standard library. In programming contests, using this file is a good idea, when you want to reduce the time wasted in doing chores; especially when your rank is time sensitive. It is basically used by testers, newbie and for education purposes. But there may arise some situation when programmers might not recall some of the header files which he/she wants to use in his program, in that situation also, this header file proves very useful. Also, there can be another case when a programmer wants to use all the standard header files except 3 or 4 header files, in that case also, this header file can make readability as well as implementation simpler.

From, software engineering perspective, it is a good idea to minimize the *'include'*. If you use this header file, it actually includes a lot of .h files, which your program may not need, thus increase both compile time and program size unnecessarily.

Though it rings programmers with lots of advantages, but basically, it holds three disadvantages as well, i.e.:

❖increases the compilation time
❖uses an internal non-standard header file of the GNU C++ library, and so will not compile in MSVC, XCode, and many other compilers
❖This header file is not part of the C++ standard, is therefore non-portable, and should be avoided

The sample code of bits/stdc++.h is:

```
// C
#ifndef _GLIBCXX_NO_ASSERT
#include <cassert>
#endif
#include <cctype>
#include <cerrno>
#include <cfloat>
#include <ciso646>
#include <climits>
#include <clocale>
#include <cmath>
#include <csetjmp>
#include <csignal>
#include <cstdarg>
#include <cstddef>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#if __cplusplus >= 201103L
#include <ccomplex>
#include <cfenv>
#include <cinttypes>
#include <cstdalign>
#include <cstdbool>
#include <cstdint>
#include <ctgmath>
#include <cwchar>
#include <cwctype>
#endif

// C++
#include <algorithm>
#include <bitset>
#include <complex>
#include <deque>
#include <exception>
#include <fstream>
#include <functional>
#include <iomanip>
#include <ios>
#include <iosfwd>
#include <iostream>
#include <istream>
#include <iterator>
#include <limits>
#include <list>
#include <locale>
#include <map>
#include <memory>
#include <new>
#include <numeric>
#include <ostream>
#include <queue>
```

```
#include <set>
#include <sstream>
#include <stack>
#include <stdexcept>
#include <streambuf>
#include <string>
#include <typeinfo>
#include <utility>
#include <valarray>
#include <vector>
#if __cplusplus >= 201103L
#include <array>
#include <atomic>
#include <chrono>
#include <condition_variable>
#include <forward_list>
#include <future>
#include <initializer_list>
#include <mutex>
#include <random>
#include <ratio>
#include <regex>
#include <scoped_allocator>
#include <system_error>
#include <thread>
#include <tuple>
#include <typeindex>
#include <type_traits>
#include <unordered_map>
#include <unordered_set>
#endif
```

It is obvious that you can make this advantageous by modifying the code and keeping those specific header files that you need for your everyday program by removing unwanted header files and then saving the file with the same name or something else along with a *.h* extension.

Again, the possibilities of increasing its advantages in cases when the programs or source codes are big and need majority of the header files, except 5 or 6 or some quantity like this. This will both reduce programmer's including of headers as well as increase readability and efficiency.

8. #exclude: A new Approach to Preprocessors Directives

A new way of eliminating unwanted header files can be introduced in the C and C++ programming world, which will exclude dynamically those header files which are not required within the program. The exclusion can also be put within condition-blocks in order to eliminate unwanted headers based on some specific conditions within the program after the program runs. And if the programmers us the bits/stdc++.h header files, there won't be any problem in eliminating those header files using the #exclude<headerFileName.h> which tells the compiler to release or exclude that specific header from the lists of headers residing inside the bits/stdc++.h code.

So the compiler developers and the C / C++ developer's community should incorporate this concept of #exclude<>. Here I'm having a sample overview kind off algorithm of how internally this #exclude changes the structure of the program, in choosing or eliminating the header file from getting called.

8.1 Rough Algorithm and explanation how will works:

```
char headerTok[100];
#if exclude "headerTok"
//Check whether <bits/stdc++.h> exists in the program
#ifdef<bits/stdc++.h> // if exists
//------------------------
 char incl[20]="#include";
 // creates a character variable storing the string "#include"
 if(headerTok == (incl+"<" +tok+ ">"))
// if the headerTok has the structure
#include<headerFileName>
 {
 filename=bits/stdc++.h;
 // open the header file bits/stdc++.h in any editor
 open(filename,"rw");
// in read-write mode, opened so that it can be dynamically
editted
 for(char word=0; word<filename.EOF; word++)
// check for the 'headerTok' word, whether matches with the
string or word inside that file
 {
 if(stringSearch(incl+"<" +headerTok+ ">") == TRUE)
// if headerTok matches with headerFIleName
 {
 incl+=("//"+incl+"<" +headerTok+ ">");
// comment that header file and save the file
 }
 }
 }
 filename.close(); // close the file
#endif
```

Let me explain what the above algorithm is doing….

Here, first you have to take a string which will hold your header file name in the form of token. Then, you have to check for the condition whether the header file name is called with the new concept and preprocessor directive *exclude*.

Then you have to check, whether your program is having the header file – bits/stdc++.h or not. If your program already include the bits/stdc++.h header then check, whether the header-file which you have mentioned along with the exclude directive is residing within the code of bits/stdc++.h or not. If yes, then the algorithm opens that bits/stdc++.h file in read-write mode, and searches for the specific string which will be the header-file name in the form of token, and put a comment i.e. "//" at the starting point of that statement. The string search will go on till it reaches the end of file. And finally the file will be saved and closed. And then in the original program, the code will execute with the include statement which will eventually exclude those header files which will be written like this:

```
#include< bits/stdc++.h >
```

```
#exclude<stdlib.h>
#exclude<math.h>
#exclude<exception.h>
# exclude<future.h>
```

This is how the entire thing will be going to work.

9. Other Recommendations and Help

I would like others to recommend me or contribute to my algorithm as to how to build the algorithm so that in case the bits/stdc++.h is used and programmers need to eliminate the header files using the #exclude<> or #exclude" "statement(s).

References

[1] Wikipedia.com
[2] Programming -- Principles and Practice Using C++ (Second Edition) – By: Bjarne Stroustrup
[3] Programming with Today's C++ (C++11 and C++14) by Bjarne Stroustrup
[4] Identifying and Understanding Header File Hotspots in C/C++ Build Processes – by: Shane McIntosh and Bram Adams

Author's Profile

Gaurav Kumar Roy has received his BCA degree from Assam University, India and then moved to Hacking and Security discipline and did Diploma in Hacking and Security under EC University, USA in 2017. Also he has done C|EH, CH|FI, VA|PT, CCNA(r&s), PMP, OWASP10 certifications. Currently he is pursuing his Masters in Computer Application (MCA) from Lovely Professional University (LPU), Punjab (with *Game Engineering and Development* as specialization). Also he is a tech writer of various tutorial-sites like: w3schools, study tonight, tutorials Cloud etc. Also he is a faculty member and a Trainer of 2 famous institutes like: CERTSTORE (certstore.in) and STUCORNER (stucorner.com). Also he was a software developer intern in ATTOCOM Pvt. Ltd. (Java-Hibernate, JDBC and RabbitMQ). He has already written *6 international research papers* in famous journals. He is the creator and host of his own YouTube channel as well.