Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

Code Golfing by Gaining and using the Intimate Knowledge of the Language

A Case Study on FizzBuzz

Deepak Ahire
Computer Science of Engineering
Walchand College of Engineering
Sangli, India

***Abstract*: Code Golfing refers to solving a problem by making use of the least number of characters i.e. by writing the shortest code as possible. Although studied extensively, the methods to solve the problem are still evolving. We generally think, "Smaller the code, the best is the performance", "Shorter code leads to a small file size and thus quicker operations, for example, downloading". But now-a-days, with use of transpilers and code minifier tools, the above arguments are redundant. A better argument is that it enhances the developer capability to think out of box and learn and use the various language features, which are less commonly used. This case study discusses the cases where the problem can be optimized by gaining and using the intimate knowledge of any language and iteratively proposes methods to devise the character optimised code of just 75 characters as final solution to the Fizzbuzz problem which is currently ranked at 4th place in the leaderboard of code-golf.io**

Keywords: C language; code golf; fizzbuzz

I. INTRODUCTION

Code Golfing refers to solving a problem by making use of the least number of characters i.e. by writing the shortest code as possible. For example, have a look at the following code snippets written in C language:

```
//Snippet A:
int main(){
   int a = 1;
   a = a + 1;
   printf("%d", a);
   return 0;
}
```

```
//Snippet B:
int main(){
   int a = 1;
   a++;
   printf("%d", a);
   return 0;
}
```

Notice that, in Snippet B, the value of the variable 'a' is incremented by 1 using the increment operator in C, thus saving 2 characters.

Although studied extensively, the methods to solve the problem are still evolving. We generally think, "Smaller the code, the best is the performance", "Shorter code leads to a small file size and thus quicker operations, for example, downloading". But now-a-days, with use of transpilers and code minifier tools, the above arguments are redundant.
A better argument is that it enhances the developer capability to think out of box and learn and use the various language features, which are less commonly used [4].
In this case study, we will discuss various cases where the problem can be optimized by gaining and using the intimate knowledge of any language.

II. METHOD

A. Problem Statement

For the case study, we will consider the most basic use case, the FizzBuzz problem.

Problem Statement: Print the numbers from 1 to 100 inclusive, each on their own line. If, however, the number is a multiple of three then print "Fizz" instead, and if the number is a multiple of five then print "Buzz". For numbers which are multiples of both three and five then print "FizzBuzz".

B. Environment used

1) **Language**: The problem can be solved using multiple languages as each language comes with its own set of features and constructs and these can be exploited to come up with a better solution, but it is unfair to compare the solutions written in different languages for the same. So, we will solve the problem using single language, C.

2) **Compiler**: Tiny C Compiler 0.9.27

3) **Compiler Flags**:
   a) **gcc -Wall option flag**: gcc -Wall enables all compiler's warning messages. This option should always be used, in order to generate better code.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

C. Methodology

In this case study, we will start off with a solution which will be in the most intelligible form, both visually and logically. In the quest to make it the better in terms of the number of characters used, we will iteratively devise a better version of code using the previous one.

III. FIZZBUZZ IMPLEMENTATION

A. Solution 1

```
#include <stdio.h>

int main(){

   for(int i=1;i<=100;i++){

      if(i%15==0){
         printf("FizzBuzz");
      }
      else if(i%3==0){
         printf("Fizz");
      }
      else if(i%5==0){
         printf("Buzz");
      }
      else{
         printf("%d", i);
      }
      printf("\n");
   }

   return 0;
}
```

Code statistics for solution 1:

1) Runtime: 0sec
2) Memory: 9.424 kB
3) # characters: 349

B. Solution 2

Features:

1) Removed the return statement in the main() function.

```
#include <stdio.h>

int main(){

   for(int i=1;i<=100;i++){

      if(i%15==0){
         printf("FizzBuzz");
      }
      else if(i%3==0){
         printf("Fizz");
      }
      else if(i%5==0){
         printf("Buzz");
      }
      else{
         printf("%d", i);
      }
      printf("\n");
   }
}
```

Code statistics for solution 2:

1) Runtime: 0sec
2) Memory: 9.424 kB
3) # characters: 335

Language intimacies learned:

1) If control reaches the end of main without encountering a return statement, the effect is that of executing “return 0;” [1].
2) C99 and C++ standards don't require functions to return a value. The missing return statement in a value-returning function will be defined (to return 0) only in the main function [2].

C. Solution 3:

Features:

1) Used Solution 2.
2) Removed the <stdio.h> header file.

```
int main(){

   for(int i=1;i<=100;i++){

      if(i%15==0){
         printf("FizzBuzz");
      }
      else if(i%3==0){
         printf("Fizz");
      }
      else if(i%5==0){
         printf("Buzz");
      }
      else{
         printf("%d", i);
      }
      printf("\n");
   }
}
```

Code statistics for solution 3:

1) Runtime: 0sec
2) Memory: 9.424 kB
3) # characters: 315

Warnings from compiler:

1) Warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

Language intimacies learned:

1) C allows you to call functions without first defining the prototypes. (C++ does not do this.) An implicit prototype for printf will be defined as follows:

   int printf();

   The calling conventions for this implicit prototype matches the actual calling conventions for printf on platform.

   "#include" doesn't import libraries, it pastes files into your source code at compile time. The "<stdio.h>" header contains (directly or indirectly) certain prototypes, but the library has to be linked in separately. Since printf() is usually in a library that is linked to programs by default, you usually don't have to do anything to use printf() [3].

D. Solution 4:

Features:

1) Used Solution 3.
2) Removed the return type for main function.

```
main(){

   for(int i=1;i<=100;i++){

      if(i%15==0){
         printf("FizzBuzz");
      }
      else if(i%3==0){
         printf("Fizz");
      }
      else if(i%5==0){
         printf("Buzz");
      }
      else{
         printf("%d", i);
      }
      printf("\n");
   }
}
```

Code statistics for solution 4:

1) Runtime: 0sec
2) Memory: 9.424 kB
3) # characters: 311

Warnings from compiler:

1) Warning: return type defaults to 'int' [-Wimplicit-int]
2) Warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]

Language intimacies learned:

1) "int" is the default data type for the main() function. If 0 is returned as exit status to indicate to the underlying platform, then it means that the program worked successfully without any compile time or run time errors. If a non-zero value (mostly 1) is returned, then it means, some error occurred in program execution.

E. Solution 5

Features:

1) Used Solution 4.
2) Integer "i" is passed to main() function as parameter.

```
main(i){

   for(;i<=100;i++){

      if(i%15==0){
         printf("FizzBuzz");
      }
      else if(i%3==0){
         printf("Fizz");
      }
      else if(i%5==0){
         printf("Buzz");
      }
      else{
         printf("%d", i);
      }
      printf("\n");
   }
}
```

Code statistics for solution 5:

1) Runtime: 0sec
2) Memory: 9.424 kB
3) # characters: 305

Warnings from compiler:

1) Warning: return type defaults to 'int' [-Wimplicit-int]
2) Warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
3) Warning: type of 'i' defaults to 'int' [-Wimplicit-int]

Language intimacies learned:

1) The main() function's argument list can be used to declare one or more variables.
2) Value of "i" is 1, as it acts as the argc or number of command line arguments. This is because argv[0] is the name of the program being executed.
3) According to Implicit Int Rule, a variable declared without an explicit type name is assumed to be of type int.

F. Solution 6A

Features:

1) Integer "i" is declared as global variable.
2) The for loop increment statement is removed and it is made the part of the condition test.

```
i;
main(){

   for(;i++<100;){

      if(i%15==0){
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

```
            printf("FizzBuzz");
        }
        else if(i%3==0){
            printf("Fizz");
        }
        else if(i%5==0){
            printf("Buzz");
        }
        else{
            printf("%d", i);
        }
        printf("\n");
    }
}
```

Code statistics for solution 6A:

1) Runtime: 0sec
2) Memory: 9.424 Kb
3) # characters: 304

Warnings from compiler:

1) Warning: return type defaults to 'int' [-Wimplicit-int]
2) Warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
3) Warning: type of 'i' defaults to 'int' [-Wimplicit-int]
4) Warning: data definition has no type or storage class

Language intimacies learned:

1) In the C language, any variable or function is implicitly int.
2) The global variables have a default value of 0.

G. Solution 6B

Features:

1) A tweak in the conditional statement is introduced which will help us design more character efficient solutions further in the study.
2) Do-while loop is used to support the syntax of the introduced tweak.

```
main(i){

    do{

        if(i%15==0){
            printf("FizzBuzz");
        }
        else if(i%3==0){
            printf("Fizz");
        }
        else if(i%5==0){
            printf("Buzz");
        }
        else{
            printf("%d", i);
        }
        printf("\n");
    }while(99/i++);
}
```

Code statistics for solution 6B:

1) Runtime: 0sec
2) Memory: 9.424 kB
3) # characters: 304

Warnings from compiler:

1) Warning: return type defaults to 'int' [-Wimplicit-int]
2) Warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
3) Warning: type of 'i' defaults to 'int' [-Wimplicit-int]

Language intimacies learned:

1) **Tweak:** "i++<100" is replaced by "99/i++" to save 1 character.
2) Let's assume A to be a non-negative and non-zero integer. Therefore, A>0. The condition (i < A), where, i>=1, can be written as ((A-1) /i). So, if A is a smallest 3 digit number, for example 100 in this case, A-1 = 99. This saves 1 character used to represent the condition.

H. Solution 7

Features:

1) The if-else ladder will be reduced by the use of ternary operator, eliminating the if-else statements and extra space and newline characters required.

```
i;
main(){

    for(;i++<100;){

printf(i%5?i%3?"%d\n":"Fizz\n":i%3?"Buzz\n":"
FizzBuzz\n",i);
    }
}
```

Code statistics for solution 7:

1) Runtime: 0sec
2) Memory: 9.424 kB
3) # characters: 109

Warnings from compiler:

1) Warning: return type defaults to 'int' [-Wimplicit-int]
2) Warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
3) Warning: type of 'i' defaults to 'int' [-Wimplicit-int]
4) Warning: data definition has no type or storage class

Language intimacies learned:

1) The if-else ladder can be reduced by making use of ternary operators.
2) This trick can be used to avoid code smells caused due to the if-else ladder.

I. Solution 8

Features:

1) Used Solution 7.
2) The printf() statement has been written inside of the for loop after the condition check.
3) Removed extra unnecessary spaces and newline characters.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

```
i;main(){for(;i++<=99;printf(i%5?i%3?"%d\n":"F
izz\n":i%3?"Buzz\n":"FizzBuzz\n",i));}
```

Code statistics for solution 8:

1) Runtime: 0sec
2) Memory: 9.424 kB
3) # characters: 84

Warnings from compiler:

1) Warning: return type defaults to 'int' [-Wimplicit-int]
2) Warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
3) Warning: type of 'i' defaults to 'int' [-Wimplicit-int]
4) Warning: data definition has no type or storage class

Language intimacies learned:

1) Every for loop is of the form:

```
for(Initialization;Condition;Afterthought)
```

Here, printf() is the part of **Afterthought** and will get executed at the end of every iteration without changing the value of "i".

2) The increment condition can be made part of **Condition**, with the use of increment operator.

J. Solution 9

Features:

1) The last 4 characters of the string "FizzBuzz", that is "Buzz" is a repetition which cost 4 extra characters in every previous solution. For this solution, use of puts() function is made to avoid unnecessary repetitions.

```
main(i){for(;100/i;puts(i++%5?"":"Buzz"))printf(i%
3?i%5?"%d":"":"Fizz",i);}
```

Code statistics for solution 9:

1) Runtime: 0sec
2) Memory: 9.424 kB
3) # characters: 75

Warnings from compiler:

1) Warning: return type defaults to 'int' [-Wimplicit-int]
2) Warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
3) Warning: type of 'i' defaults to 'int' [-Wimplicit-int]
4) Warning: implicit declaration of function 'puts' [-Wimplicit-function-declaration]

Language intimacies learned:

1) Optimising code by avoiding repetitions in the code.
2) puts() can be used instead of printf() to print the strings.

NOTE: This solution is currently ranked 4th on the code-golf.io for the FizzBuzz problem.

IV. CONCLUSION

In this case study, we discussed how the FizzBuzz problem can be optimised by gaining and using the intimate knowledge of any language. Recruiters can use this test to filter out strong candidates for language specific roles as it not only teaches us the best practices to minimise the code, but it is also useful in highlighting the wrong practices. The compiler designers can use this test to learn the shortcomings of the existing languages to develop new languages. The tweaks discussed in the solutions can also be used in designing kernels for operating systems and data intensive operations like real time audio and video streaming. Therefore, code golfing can act as one of the best programming language tests.

ACKNOWLEDGMENT

I am sincerely thankful to the CodeGolf Stack Exchange community for providing more insights into the problem in the form of answers and discussions.
I also thank the code-golf.io community for promoting code golfing and making the platform open source for contribution.

REFERENCES

[1] Tamás Szelei (2013) Why does the main function work with no return value?, Available at: https://stackoverflow.com/questions/19293642/why-does-the-main-function-work-with-no-return-value (Accessed: 27th May 2020).
[2] fnieto - Fernando Nieto (2017) Why does flowing off the end of a non-void function without returning a value not produce a compiler error?, Available at: https://stackoverflow.com/questions/1610030/why-does-flowing-off-the-end-of-a-non-void-function-without-returning-a-value-no (Accessed: 27th May 2020).
[3] Dietrich Epp (2012) Using printf function without actually importing stdio.h and it worked?! Why is that so? [duplicate], Available at: https://stackoverflow.com/questions/11150883/using-printf-function-without-actually-importing-stdio-h-and-it-worked-why-is (Accessed: 27th May 2020).
[4] Deepak Ahire (2020) 'The Journal of Codegolfing', Journal of Brief Ideas, [Online].Available at: http://beta.briefideas.org/ideas/159238dbf448802c3a17520aa18b3e87 (Accessed: 28th May 2020).

(This work is licensed under a Creative Commons Attribution 4.0 International License.)