C Language Source Code Adornment Project

By Md. Danish Anwar

December 2017

Contents

1 Abstract

Most of the programmers starts their journey with C language. As a beginner when we write the code for a given problem, the code is not always in the standard ANSI C format. Two different programmer can have different format for the same code. So for sake of recognition standard ANSI C format is chosen.The goal of our project is to convert any compilable C source code to it's standard ANSI C format using LEX and YACC.

2 Introduction

XYZ is a software development tool that converts C source file to it's equivalent standard ANSI C format.As a beginner, our code in not always properly indented and sometimes we spend more time in indenting the code rather than writing it. Same happened with me as well. I always wanted to build something which can properly indent my code so that I will not have to worry about the indentation.Although there are quite a few resources available but I was not satisfied at all thus I started building XYZ which is indeed very useful for beginners as well as for proficient ones.

This paper provides a brief overview of XYZ with quite a few examples and mentioning all the techniques used.

3 ANSI C format

In ANSI C format, the code is properly indented and it removes all the redundant spaces in our code.Only one variable is declared in one line

3.1 Example of Declaration

C source code format(input)

```
int a,  b = 4, c=8e5;
```

Standard ANSI C format(output)

```
int a;
int b = 4;
int c=8e5;
```

3.2 Example of Indentation

C source code format(input)

```
for(i=0;i<3;   i++){
d=d+e;
}
```

Standard ANSI C format(output)

```
for( i = 0 ; i < 3 ; i++)
{
    d=d+e;
}
```

3.3 Example of Sample Source Code

C source code format(input)

```
#include  <stdio.h>
#include<stdlib.h>
int main(){
while(t--){
for(i=0;i<n;i++){
        while(k--){
while(j--){
a=b;
}
}
}
while(a>b){

a++;
}


do{a++;
}while(t--);









}
}
```

Standard ANSI C format(output)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    while( t--  )
    {
        for( i = 0 ; i < n; i++  )
        {
            while( k--  )
            {
                while( j--  )
                {
                    a = b;
                }
            }
        }
        while( a > b )
        {
            a++ ;
        }
        do
        {
            a++ ;
        }
        while( t--  );
    }
}
```

4 Tools Used

In this project we have used..

- C language
- Lex
- Yacc

5 Overview

There are two files in this tool.One is .l(lex file) another is .y(yacc file) In lex file the regular expression for all combination of possible words and expressions of source code is written. A token is assigned to every regular expression. When yacc parses the source code, it recieves the tokens generated in lex file. YACC uses context free grammar for parsing. The context free grammar for all the tokens is written in YACC file. It parses the source code using the grammar and evaluates the code for indenting source file.

5.1 Regular Expressions

5.1.1 Header

C source code will contain header files on the top. During parsing headers files will be parsed first. Header file can be of two types

- predefined header file
- User-defined header file

The predefined header file can be of type

```
#include<abc.h>
```

So it's regular expression can be written as

```
preheader   [#][ ]*include[ ]*[\<][ ]*[a-z]+[\.][h][ ]*[\>]
```

The user-defined header file can be of type

```
#include"abc.h"
```

So it's regular expression can be written as

```
userheader  [#][ ]*include[ ]*[\"][ ]*[a-z]+[\.][h][ ]*[\"]
```

5.1.2 Datatype

Data types can be float, int or char or static int etc. So regular expression for data types can be written as

```
(static|const|unsigned|long|[ ]*)*[ ]*(int|float|char|double)
```

5.1.3 Variable Name

In C variable name can start with any alphabet(both upper and lower case) or underscore.It can not start with a numerical value though it mau contain any numerical value afterwards along with any alphabet and any numbers of underscores. So regular expression for variable name can be written as

```
[a-zA-Z_][a-zA-Z0-9_]*
```

5.1.4 Variable Value

The value of any variable can be integer or float or exponential. So regular expression for variable name can be written as

```
[0-9][0-9\.Ee]*
```

5.1.5 Function Prototype

The function prototype can be of the form , return data type of the function followed by it's name and arguments. The regular expression for argument inside brackets can be given as

```
([\(][ ]*({datatype}[ ]*{name}[ ]*[,]?[ ]*)+[\)])|([\(][ ]*[\)])
```

5.2 Tokens

For every regular expression in lex file we will assign a token. First we will declare an empty string and than store the input in that string omitting all redundant spaces in between Declaring empty string

```
%{
char TOKEN_STRING[200];
%}
```

Skipping all spaces , tab and newline

```
[\t \n] ;
```

5.2.1 Header

We will return a token HEADER for both predifined header and user-defined header. Code for it is given below

```
{preheader}  {
               int j = 0;
               int i = 0;
               for( i = 0;i < strlen(yytext); i++ )
               {
                  if(yytext[i]!=' ')
                  TOKEN_STRING[j++]=yytext[i];
          }
               TOKEN_STRING[j]='\0';
               yylval.sval=TOKEN_STRING;
                  return  HEADER;
      }

{userheader} {
```

```
        int j = 0;
        int i = 0;
        for( i = 0;i < strlen(yytext); i++ )
        {
            if(yytext[i]!=' ')
            TOKEN_STRING[j++]=yytext[i];
        }
        TOKEN_STRING[j]='\0';
        yylval.sval=TOKEN_STRING;
           return  HEADER;
    }
```

The grammar for this token HEADER is written in yacc file. This code basically omits all extra spaces and tabs from user's input (source code) and converts the header file to it's standard ANSI C format.

5.2.2 Data Type

We will be returning a token DATATYPE for data type. It will basically omit extra spaces before function declaration and will store the standard output in the string.Code for this is as follows

```
{
  int j = 0;
  int i = 0;
  int flag = 1;
  for(i = 0;i < strlen(yytext); i++)
  {
    if(yytext[i]==' '&&flag==1)
    {
    TOKEN_STRING[j++]=yytext[i];
    flag = 0;
  }
  else if(yytext[i]!=' ')
  {
    TOKEN_STRING[j++]=yytext[i];
    flag = 1;
  }
  TOKEN_STRING[j] = '\0';
  yylval.sval = TOKEN_STRING;
    return DATATYPE;
}
```

5.2.3 Function argument

After data type there will be name of the function followed by function argument(if any). We will print the name of the function as given without making any changes to it. In the argument we will remove extra spaces and convert it to standard ANSI C format and store it in a string. After that we will return a token named ARGUMENT for function argument. Code for this is as follows :

```
{
                    int j=0;
                    int i=0;
                    int flag = 1;
                    for(i = 0;i < strlen(yytext); i++)
                    {
                       if(yytext[i]==' '&&flag==1)
                        {
                            TOKEN_STRING[j++]=yytext[i];
                            flag = 0;
                        }
                        else if(yytext[i]!=' ' && yytext[i]!=',')
                        {
                        if(yytext[i]!='(' && yytext[i]!= ')')
                        {
                            TOKEN_STRING[j++]=yytext[i];
                            flag = 1;
                        }
                        else if(yytext[i] == '(')
                        {
                            TOKEN_STRING[j++] = '(';
                         TOKEN_STRING[j++] = ' ';
                         flag = 0;
                        }
                        else if(yytext[i]==')')
                        {
                         if(yytext[i-1]!=' ')
                            TOKEN_STRING[j++]=' ';
                         TOKEN_STRING[j++] = ')';
                         flag = 0;
                        }
                        else if(yytext[i]==',')
                        {
                            if(yytext[i-1]!=' ')
                                TOKEN_STRING[j++]=' ';
                            TOKEN_STRING[j++]=',';
                            TOKEN_STRING[j++]=' ';
                            flag = 0;
                            }
```

```
        TOKEN_STRING[j] = '\0';
        yylval.sval = TOKEN_STRING;
        return ARGUMENT;
    }
```

5.2.4 Main Function

After declaring all the functions there should be main function and main will also be parsed as above. Inside the function main there will be body (like loops,switch , cases, statements etc). We will be indenting these parts of body as well and will give them token.

5.2.5 For Loop

Inside the body whenever we encounter a for loop we will run the below code and return token FOR to yacc which will parse the token and will evaluate the code by traversing the grammar Code :

```
{
   strcpy(TOKEN_STRING, yytext);
   yylval.sval = TOKEN_STRING;
   return FOR;
}
```

5.2.6 While, if and else if

While if and else if blocks are similar in syntax so for each of them we will copy the name of block in a string and return a token BLOCK_TYPE to yacc. Yacc will parse the token and evaluate the code by traversing the grammar

```
{
  strcpy(TOKEN_STRING,yytext);
  yylval.sval = TOKEN_STRING;
  return BLOCK_TYPE;
}
```

5.2.7 Switch and Case

Similarly for switch and case we will return a token named SWITCH and CASE respectively. Yacc will do the parsing and traversing of grammar.

```
    {
      return SWITCH;
    }

    {
      return CASE;
    }
```

5.2.8 Variable name and value

We will not do any changes to the variable name given by the user and store the name of variable in a string. We will return a token named NAME.

```
{
    strcpy(TOKEN_STRING, yytext);
    yylval.sval = TOKEN_STRING;
    return NAME;
}
```

We will do the same thing for variable value and return a token named NUMBER.

```
{
    yylval.ival = atoi(yytext);
    return NAME;
}
```

5.2.9 Operators

For operators of unit length we do not have to worry about. Those will be directly parsed by yacc but operators with length greater than 1, yacc will be unable to parse it directly. We will have to return a token for each operator having length greater than 1. Yacc will than parse that token.

```
"<="            return LE;
">="            return GE;
"=="            return EQ;
"!="            return NE;
"||"            return OR;
"&&"            return AND;
```

So first header, than function data type followed by function argument will be parsed. Than the remaining body containing blocks of loops and switch cases will be parsed. So in this way whole source code will be parsed to yacc.

ASSUMPTION We are assuming that all variables are declared locally and there is no scope of declaring variables globally.

6 Context Free Grammar for Parsing

We will be using context free grammar for parsing our source code. Context free grammar is defined by four tuples i.e. Non terminals, set of alphabets, start symbol and production rules. Our grammar G is defined as

$$G = \{V, \Sigma, S, P\}$$

where V is Set of non terminals given by

$$V = (,\ \ ;\ \ :\ \ \{\ \ \}\ \ S\ \ H\ \ F\ \ B\ \ I\ \ D\ \ A\ \ C\ \ T)$$

Σ is set of alphabets given by

$$\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q\}$$

S is the start symbol.
P is the production rule given by,

$$A \rightarrow \alpha$$

where A $\in$ V and $\alpha \in$ (V $\cup \Sigma$)*
In our grammar the production rules are as follows...

$$S \rightarrow H|\phi$$
$$H \rightarrow a\ S\ |b\ i\ c\ F$$
$$F \rightarrow ;\ S\ |B\ S$$
$$B \rightarrow \{\ B\ \}$$
$$I \rightarrow b\ D\ I\ \ |\ i\ n\ I\ |\ d\ o\ B\ I\ |\ f\ n\ B\ I\ |\ e\ f\ n\ I\ |\ l\ b\ I\ |\ j\ I\ |\ g\ n\ A\ I\ |\ q\ I\ |\ \phi$$
$$A \rightarrow \{\ C\ \}$$
$$C \rightarrow h\ m\ I\ C\ |\ k\ I\ |\ \phi$$
$$D \rightarrow i\ T\ |\ i\ =\ m\ T$$

$$T \rightarrow ,\ D\ |\ ;$$

6.1 Explanation

Our source code is assumed to be divided into three major blocks.These are

6.1.1 Header Block

This block includes all the declaration of header files. This is a separate block and does not need any indentation. It should be at the top of the code with no preceding spaces. But the user may give redundant spaces in the declaration of header files, so our tool will remove the redundant spaces by modifying the tokens given by lex files.

6.1.2 Function Prototype Block

The function prototype block contains all the prototype of user defined functions with data type, name of the function, argument of the function followed by a semicolon. This block does not need any indentation. It should be at the top of the code with zero preceding spaces. But the user may give redundant spaces in between the arguments, so our tool will remove the redundant spaces by modifying the tokens given by lex files.

6.1.3 Function Block

The function block will all the function and the statement inside the function. The fuction the block is similar to the function prototype block with respect to data type name and argument of the function. But after arguments it is followed by an opening curly bracket followed by body of the code and closing bracket. This block requires indentation as it contains opening and closing braces. So to acconplish that task we define global variable "space" and initialize it with zero in our .y file. Now whenever we encounter an opening bracket we will print the opening bracket in new line and than we will increament the value of space by 4. Now we will parse the remaining code and in the begining of every new line we will print say "x" no of spaces where "x" is the current value of space variable. Now whenever we encounter a closing bracket we will print the closing bracket in new line and than we will decreament the value of space by 4.Now we will parse the remaining code and in the begining of every new line we will print say "x" no of spaces where "x" is the current value of space variable. Between the opening and closing braces we may encounter several blocks like "for", "while", "else", "else if", "do while", "switch case" and so on. The language for indenting these blocks has given in our grammar.

6.1.4 Tokens and Alphabets assigned to them.

Tokens Alphabet assigned to that token

HEADER	a
DATATYPE	b
ARGUMENT	c
FOR	d
DO	e
BLOCK_TYPE	f
SWITCH	g
CASE	h
NAME	i
BREAK	j
DEFAULT	k
ELSE	l
NUMBER	m
CALL_ARGUMENT	n
FOR_ARGUMENT	o
BLOCK_ARGUMENT	p
OTHER	q

6.1.5 Non-terminals used in grammar and their meaning.

Non-terminals Used Their Meaning

Non-terminals used	Meaning
S	START_SYMBOL
H	HEAD
F	FUNC_TYPE
B	BODY
I	INNERBODY
D	DECL_TYPE
A	SWITCH_BODY
C	SWITCH_INNERBODY
T	TERMINATE
,	,
;	;
{	{
}	}

6.2 Challenging Issue

The main challenge while indenting the source code was to print every declaration on a new line. User may declare many number of variables in a single line separated by comma. We have to print every variable on new line with it's data type and name followed by a semicolon. Code for this as follows...

```
    INNERBODY : DATATYPE {strcpy(var_datatype,$1);}
    DECL_TYPE
INNERBODY
DECL_TYPE : NAME {give_t_space(); printf("%s %s ;\n",var_datatype,$1);}
TERMINATE
| NAME {give_t_space(); printf("%s %s ",var_datatype,$1);}
'=' NUMBER {printf("= %s;\n",$1);}
TERMINATE
;
TERMINATE : ',' DECL_TYPE
| ';'
;
```

7 Running the Tool

To run this tool first one should have flex and bison installed on their machine in order to run lex and yacc files.

7.1 Installation(On Linux)

To install flex

```
sudo apt-get install flex
```

To install bison

```
sudo apt-get install bison
```

7.2 Running and compiling lex and yacc files

After installing flex and bison we will have to run two files with the following commands

```
lex calc.l
yacc calc.y
gcc y.tab.c -ll -ly
```

after running the last command an executable file a.out will be generated. We will have to give input source code(say test.c) to the a.out file and than run it with following command.

```
./a.out <test.c
```

After executing this command our properly indented C code will be generated as output which is the ANSI C format.

8 Final Output

So far whatever we have done will convert even a poorly written compilable C source code to it's equivalent Standard ANSI C format.
Following are some examples of our final output for some sample codes.

8.1 Example-1

8.1.1 Input C Source Code

```
#include <stdio.h>
int main()
{int year;
printf("Enter a year: ");
    scanf("%d",&year);
        if(year%4 == 0)
    {
        if( year%100 == 0)
{
    if     ( year%400 == 0){
     printf("%d is a leap year.", year);
    }
        else{
   printf("%d is not a leap year.", year); }
        }else{
            printf("%d is a leap year.", year );
        }}
else{
        printf("%d is not a leap year.", year);
                                                    }
}
```

8.1.2 Output in Standard ANSI C format

```
#include<stdio.h>
int main(  )
{
    int year ;
    printf("Enter a year: ") ;
    scanf("%d",&year) ;
    if(year%4 == 0)
    {
        if( year%100 == 0)
        {
            if( year%400 == 0)
            {
                printf("%d is a leap year.", year) ;
            }
            else
            {
                printf("%d is not a leap year.", year) ;
            }
        }
        else
        {
            printf("%d is a leap year.", year ) ;
        }
    }
    else
    {
        printf("%d is not a leap year.", year) ;
    }
}
```

8.2 Example-2

8.2.1 Input C Source Code

```
#   include   <stdio.h>
int  fun   ( int a,int b   ,   int c     )  ;
float   anotherFun (int a);
unsigned long     long    int    diffFun(  int    a,           int b,  int c    )
              {int a = 4,c;
             printf("%s",a);
             diffFun(6,9,2);
             int a = 5,b = 7;
             for(i=0;i<20&&i>1;i++)
             {
              anotherFun(5);
             int a = 5,b = 7;
             for(j=0;j<n||k>2;j=j+2)
             {
              int a,b;

              if(i<100&&x==3)
             {
              int a,b,c;
                        switch (x)
{
case 1:
        printf("Choice is 1");
break;
case 2:
        printf("Choice is 2");
        printf("hello");
case 3: printf("Choice is 3");
break;
default: printf("Choice other than 1, 2 and 3");
break;
}
             }
             else    if(a==0)
             {
              int  b;
              do {
              int a,b=9;
             } while(a==0&&b==9);
             while(t--)
             {
              int a=0;
```

```
                }
                }
                else
                {
                 int p = 0;
                }
                }
                }
                if(i<100&&x==3)
                {
                 int a,b,c;
                }
                else    if(a==0)
                {
                 int  b;
                }
                else
                {
                 int p = 0;
                }
                do {
                 int a,b=9;
                } while(a==0&&b==9);
                while(t--)
                {
                 int a=0;
                }
}
```

8.2.2 Output in Standard ANSI C format

```
#include<stdio.h>
int fun( int a , int b , int c );
float anotherFun( int a );
unsigned long long int diffFun( int a , int b , int c )
{
    int a = 4;
    int c ;
    printf("%s",a) ;
    diffFun(6,9,2) ;
    int a = 5;
    int b = 7;
    for(i=0;i<20&&i>1;i++)
    {
        anotherFun(5) ;
        int a = 5;
        int b = 7;
        for(j=0;j<n||k>2;j=j+2)
        {
            int a ;
            int b ;
            if(i<100&&x==3)
            {
                int a ;
                int b ;
                int c ;
                switch(x)
                {
                    case 1 :
                        printf("Choice is 1") ;
                        break ;
                    case 2 :
                        printf("Choice is 2") ;
                        printf("hello") ;
                    case 3 :
                        printf("Choice is 3") ;
                        break ;
                    default :
                        printf("Choice other than 1, 2 and 3") ;
                        break ;
                }
            }
            else if(a==0)
            {
                int b ;
```

```
                do
                {
                    int a ;
                    int b = 9;
                }
                while(a==0&&b==9) ;
                while(t--)
                {
                    int a = 0;
                }
            }
            else
            {
                int p = 0;
            }
        }
    }
    if(i<100&&x==3)
    {
        int a ;
        int b ;
        int c ;
    }
    else if(a==0)
    {
        int b ;
    }
    else
    {
        int p = 0;
    }
    do
    {
        int a ;
        int b = 9;
    }
    while(a==0&&b==9) ;
    while(t--)
    {
        int a = 0;
    }
}
```

9 Acknowledgment

The author would like to thank Prof. J. Howlader for the technical support. This project was completed by Md. Danish Anwar, under the supervision of Prof. J. Howlader.

10 References

[1] Lex Yacc, 2nd Edition by O'Reilly.

Let Us C by Yashavant Kanetkar (Author) 2016 Edition

11 About The Author

MD Danish Anwar is currently an undergraduate student in NIT Durgapur. This project was completed in December 2017. Born in 1998, in Kolkata, Danish is currently persuing B.Tech in Computer Science.