DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

(Session : January – May 2016)

PROJECT REPORT

ON

VARIABLE NAME REPLACEMENT

Submitted by

1.	Rishabh Soni	01FB14ECS189
2.	Satyam Bharadwaj	01FB14ECS214
3.	Siddharth Srinivasan	01FB14ECS241

Program	:	B. Tech - CSE
Semester	:	IV
Section	:	D
Group No	:	22

DESIGN AND ANALYSIS OF ALGORITHMS

REPORT TITLE

VARIABLE NAME REPLACEMENT IN A PROGRAM

Introduction

Variables in a program are usually named based on the user's preference or based on their use , this is troublesome for other teams working on the project as language or words may vary with programmers. The solution to this problem is our program. Our program inputs a file and it lets the user replace each variable declaration with name of their own choice. The change of name will help them understand the program better and it also saves time . All he has to do is change the names of the variables of the concerning program to another name that he so desires.

Our project is made while testing with C programs, so the renaming will work with C/C++ programs only. And we have restricted the renaming part of the variables to only primitive data types such as int,float,char,double etc. Our program even handles the renaming of arrays, pointers and other such variables of primitive types. We assume that whatever the input program that the end user sends to our program is syntactically correct and of a reasonable working condition.

The data structures used in our project are stacks, linked lists, arrays, structures etc. We have tried to implement modularity by writing methods/functions to perform repetitive procedures.

Working

- Input : File containing the code is opened in read mode.
- Create Symbol Table : For generating name and position links
- Acquiring new replacement names
- Check occurrences and addresses of the variables
- Check if the user has used a legal identifier name
- Replace the old name with the new name

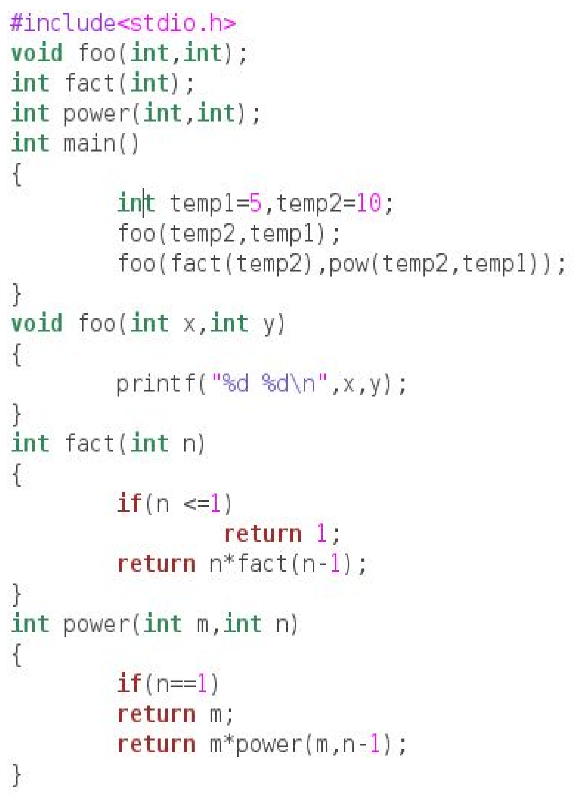
The symbol table that our program generates uses **structures** and has the following three useful attributes:-

1. Variable Name
2. Primitive data type of variable and
3. Addresses of occurrences

Apart from this, various other cases were accounted for while generating an ideal symbol table. For instance, the scope of the various variables. For dealing with the three types of scopes of a typical C program (local scope, global scope and function scope) we have made use of a **stack** which implements the functioning of parenthesis matching. Another problem to deal with was if any variable present in the program was a substring of another variable (or even a function). In this case, the output would consist of new variable names that the user has not even prompted for. To deal with this, checks were made in the program to eliminate such problems.

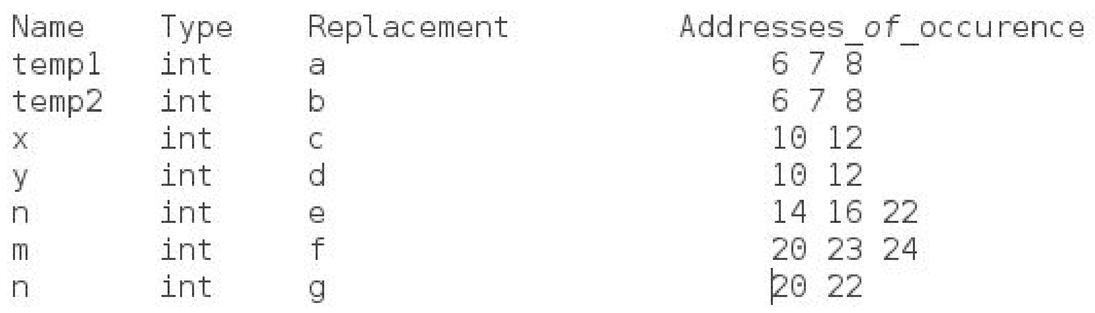
After this comes the prompt for replacing the old names with new ones. We first check whether the names received from the user follow the rules of naming identifiers of a typical C/C++ language. Once we obtain the new names from the user which follows the required conventions we apply a sequence of various **brute-force** checks and replacement techniques on the input file and generate a new output file with the required names of the variables that is needed by the user.

Figure 1 . This Image is a test code for the program.

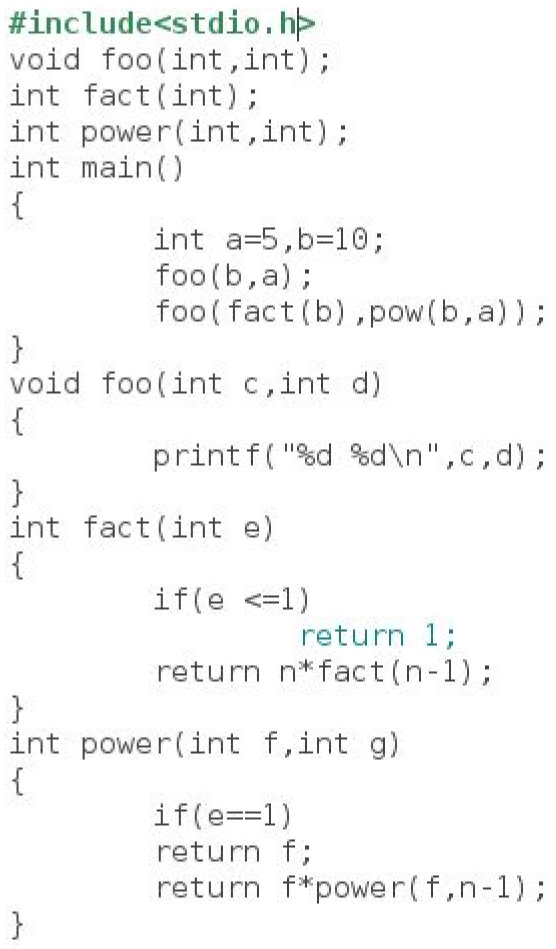


```
#include<stdio.h>
void foo(int,int);
int fact(int);
int power(int,int);
int main()
{
        int temp1=5,temp2=10;
        foo(temp2,temp1);
        foo(fact(temp2),pow(temp2,temp1));
}
void foo(int x,int y)
{
        printf("%d %d\n",x,y);
}
int fact(int n)
{
        if(n <=1)
                return 1;
        return n*fact(n-1);
}
int power(int m,int n)
{
        if(n==1)
        return m;
        return m*power(m,n-1);
}
```

Figure 2: Represents the Symbol Table for the above code.



```
Name     Type     Replacement          Addresses_of_occurence
temp1    int      a                         6 7 8
temp2    int      b                         6 7 8
x        int      c                         10 12
y        int      d                         10 12
n        int      e                         14 16 22
m        int      f                         20 23 24
n        int      g                         20 22
```



```
#include<stdio.h>
void foo(int,int);
int fact(int);
int power(int,int);
int main()
{
        int a=5,b=10;
        foo(b,a);
        foo(fact(b),pow(b,a));
}
void foo(int c,int d)
{
        printf("%d %d\n",c,d);
}
int fact(int e)
{
        if(e <=1)
                return 1;
        return n*fact(n-1);
}
int power(int f,int g)
{
        if(e==1)
        return f;
        return f*power(f,n-1);
}
```

Figure 3: Represents the Code after replacement

There are quite a few advantages of this program. It is particularly useful when there is a large program, one can easily replace the names throughout the program effortlessly

We would like to thank Mr. N.S. Kumar for providing us with the idea to implement the project and his constructive criticism throughout.