CSCB09 Assignment 2 Documentation- SimpleMemModAnalyzer.c

THE PROBLEM

Build our own simplified version of a memory model --*address space*-- visualizer.

THE SOLUTION

My approach to a problem of this magnitude was to split the program into smaller parts.

These parts were:

- Static Data
  - Store all the global variables found in the program
  - All global variables will be declared at the top
- Heap Data
  - Store all variables with space allocated by malloc/calloc
- Stack Data
  - Store all other variables that are declared inside functions
- ROData
  - Store all the string literals found in the program
- Stats
  - Find total number of lines in program
  - Total number of functions
    - Total number of lines in function
    - Total number of variables in function
- Command Line Arguments
  - Only takes one argument, which is the program we want to analyze
  - Throw error to stderr if too many or too few arguments
  - Throw error to stderr if not a valid file

LIBRARIES USED

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define MAX_LENGTH 256
```

STRUCTS

```
typedef struct variables{
    char *name;
    char *scope;
    char *type;
    int size;
    char *data_size;

    int num_lines_function;
    int num_var_function;

    int heap_stack; //0 if global, 1 if heap, 2 if stack, 3 if ROData

    struct variables *next;

}VARNode;

//initliaze VARNode
VARNode *initialize_var(char *name, char *scope, char *type, int size, char
*data_size, int lines, int variables, int heap_stack){

//insert into VARNode linked list
VARNode *insert_var(VARNode *head, char *name, char *scope, char *type, int size,
char *data_size, int lines, int variables, int heap_stack){
```

- Used for storing global variables and variables inside functions

```
typedef struct functions{
    char *name;
    int total_lines;
    int total_variables;
    struct functions *next;
}FUNNode;

//intitalize FUNNode
FUNNode *intialize_fun(char *name, int total_lines, int total_variables){

//insert into FUNNode linked list
FUNNode *insert_fun(FUNNode *head, char *name, int lines, int variables){
```

- Used for storing functions

FUNCTIONS

--------------MISCELLANEOUS--------------

```c
//gets rid of any comments made in the line
char *cleanline(char *line){
```

```c
//returns the line with all the whitespace removed
char *remove_whitespace(char *line){
```

```c
//use to count the number of lines in the program/function
//use "" as fun_name when counting lines of entire program
int line_counter(FILE *fp, char *path, char *fun_name){
```

```c
//returns 1 if line is for loop, 2 if while loop, and 0 otherwise
int check_loop(char *line){
```

```c
//returns 1 if line is if/else statement, and 0 otherwise
int check_if_else(char *line){
```

```c
//returns 1 if line is return or printf, and 0 otherwise
int check_return(char *line){
```

```c
//gets the number of variables in the line
int num_variables_line(char *line){
```

```c
//stops the line after encountering either a ";" or a "=" or a ")"
char *remove_equal_colon_bracket(char *line){
```

```c
//returns the line with all the colons/stars/whitespace removed
char *remove_stars(char *line){
```

```c
//removes the last bracket in line
char *remove_last_bracket(char *line){
```

```c
//return 1 if line contains malloc or calloc, return 2 otherwise
int is_heap_RO(char *line){
```

```c
//return 1 if line contains malloc or calloc
int check_alloc(char *line){
```

```c
//return 1 if line contains ; and =
int check_equal_colon(char *line){
```

--------------GLOBAL VARIABLES--------------

```c
//finds the global variables of the program
//assumption is that global variables are all declared at the starting of the program
VARNode *find_global_var(FILE *fp, char *path, VARNode *head){
```

--------------HELPER FUNCTIONS FOR ALL VARIABLES--------------

```c
//returns the datatype of the variable in the line
char *get_datatype(char *line){
```

```c
//returns the value inside the bracket, given that the bracket is not empty
int value_of_bracket(char *line){
```

```c
//returns the size of the datatype given in string format, in either int ot char* form
char *get_sizeof(char *datatype, int *size){
```

```c
//returns the stuff inside brackets
//if option is 1, then malloc
//option 2 is calloc, anything else is [] (for example, option = 3)
char *get_inside_bracket(char *line, int option){
```

```c
//returns 1 if line is a function header, 2 if a variable, 0 otherwise
int check_function_datatype(char *line){
```

--------------FUNCTION HEADER VARIABLES--------------

```c
//gets the datatype of the header, given there is only one variable in the header.
char *get_header_datatype(char *function_header){
```

```c
//adds the header variables to the linked list
```

```c
VARNode *print_header_variables(char *function_header, char *fun_name, int
variables, VARNode *var_head){
```

```c
//counts the number of variables in the header
int num_variables_header(char *function_header){
```

--------------FUNCTION VARIABLES--------------

```c
//goes through each function and adds the variables inside it to the linked list
VARNode *analyze_function(FILE *fp, char *function_header, VARNode *var_head,
VARNode *global_head){
```

--------------VARIABLES RE-DEFINED--------------

```c
//adds the variables that have had their values updated (originally global, or
changed to malloc/calloc)
//to the linked list
void add_updated_var(char* name, VARNode *var_head, VARNode *global_head, char
*fun_name, int check_heap, char *line){
```

--------------OUTPUT FUNCTIONS--------------

```c
//returns 1 if name already belongs to the linked list
//used in creating the FUNNode linked list of functions
int if_belongs(FUNNode *head, char *name){
```

```c
//prints the line to stdout, choosing whether to have size as int or string
void print_correct_line(VARNode *head){
```

```c
//the controller function of entire program
//prints out all the required data, based on where it should be printed
void analyze_source(FILE *fp, char *path){
```

--------------CLA FUNCTIONS--------------

```c
//returns 1 if valid file, 0 if not
//https://stackoverflow.com/questions/230062/whats-the-best-way-to-check-if-a-
file-exists-in-c
int valid_file(char *path)
```

```
//Determines if command line arguments are valid
int main(int argc , char **argv){
```

HOW TO RUN

compile the program: gcc SimpleMemModAnalyzer.c

Specify the program you want to analyze

- You may only specify one program, and it must be valid, otherwise you will get an error

REPORT:

- Any functions with multiline comments may give wrong output. Single line comments are fine. So, don’t use any functions with multiline comments
- If datatype if any variation of int[]/float[]/char[], the datatype printed would be int[]/float[]/char[].
    - ex. int a[5], the datatype is printed as int[], and size printed as 5