Yeditepe University
Department of Computer Engineering

CSE 232
Systems Programming
Fall 2023

Term Project

Debugger/Preprocessor

Due to: 5th January 2024

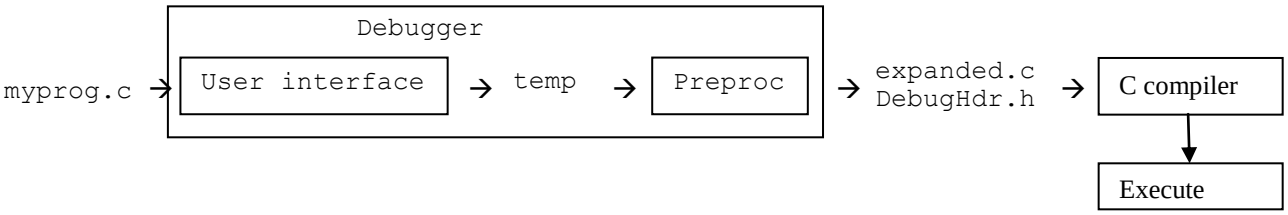
4 Students in a Group

In this project, you will develop a **debugger** for C language. Your debugger will consist of some debugger functions which will be inserted in the user programs using a **preprocessor** that you will write.

Write your Debugger/Preprocessor in C and use gcc compiler on Linux.

The debugger consists of 2 modules:
- User interface: to insert debugger directives in a C program
- Preprocessor: to substitute C functions in the place of the debugger directives

Structure of the project is given below.



Debugger

In order to debug a program some debugger directives must be inserted in the program. In this project following two directives must be implemented:

`var:` defines the variable to be traced
`trace:` displays the execution trace of a variable

Ex:

myprog.c

```
#include "DebugHdr.h"
int main()
{
      int a, b;
      b=0;
      a=1;
      while(b<10)
      {
            a=a+b;
            b=b+1;
      }
      return 0;
}
```

If the user wishes to trace variables `a` and `b`, he/she inserts `var` directive for each variable to be traced; and `trace` directives to the lines where their values should be displayed.

```
#include "DebugHdr.h"
int main()
{
	int a, b;
	@var a // variable "a" will be traced
	@var b // variable "b" will be traced
	b=0;
	a=1;
	while(b<10)
	{
		a=a+b;
		b=b+1;
		@trace b // display the last value of variable "b"
	}
	@trace a // display last value of variable "a"
	return 0;
}
```

User Interface

You must write a user interface for the user to insert the debugger directives easily. The user interface must display the program on the screen and allow the user to insert `@var` and `@trace` lines in the program using the cursor. Following interface commands must be implemented:

- `d:` displays the program on the screen
- `w:` moves cursor up
- `s:` moves cursor down
- `i:` inserts a `var` or a `trace` directive in the program to the current position of the cursor
- `x:` exit from the user interface and continue with the preprocessor

Use NCURSES library to move the cursor.

When the interface is started, first, it must read the program (myprog.c) from the file and store it in a buffer as a linked list structure and also display it on the screen. You must use the following structure for the buffer. Assume that there are maximum 100 lines of code and each line is maximum 80 characters. `link` points to the index of the next line in the `Buffer[]` array.

```
struct buf {
	char line[80];	// a program line
	int link;	// link to the next line
}
struct buf Buffer[100]; // max 100 lines
int head;		//head node
```

You can declare `Buffer[]` as global.

Then, using `w` and `s` keys, the user moves the cursor onto the line that the debugger directive will be inserted. Then the user presses `i` key in order to insert the directive. When `i` key is pressed, the cursor moves to the last line on the screen and the user writes a `var` or a `trace` directive. This new line must be added to the end of the buffer and the links must be updated. After all the directives are inserted in the code, the user presses `x` key to exit from the interface and proceed with the preprocessing stage. It writes the `Buffer[]` to the file `temp` .

Write the following interface functions:

`read_to_buffer()`: reads original C code (myprog.c) into `Buffer[]`.
`add_to_buffer()`: inputs the debugger directive typed by the user. Adds it to `Buffer[]` and updates the links
`write_to_file()`: writes the buffer to the file temp

Preprocessor

The preprocessor first reads the file temp into the `Buffer[]` array. Then it goes through the code in the `Buffer[]` and expands it by substituting the related C function calls in the places where debugger directives exist. At the end, it writes the expanded code to a file named `expanded.c`. For this purpose, the preprocessor follows the links in the `Buffer[]` array and for each line, it checks if that line contains a debugger directive or not (Debugger directives start with `@`). If the line contains a debugger directive, the preprocessor substitutes the related function call instead of the debugger directive.

The preprocessor needs to store the names of the variables to be traced in the `Tr_var[]` array:

```
char Tr_var[20];  // assume that each variable name is 1 character
int nTrvar;       // number of variables in the array
```

Your preprocessor must do the following substitutions:

1. Processing `var` directive
   get the variable name from the directive and replace the line with the call to `add_TT()`
   add variable name to `Tr_var[]` array
   Ex: `@var a` → `add_TT('a', a);`

2. Processing `trace` directive
   get the variable name from the directive and replace the line with the call to `display_TT()`
   Ex: `@trace a` → `display_TT('a');`

3. Processing assignment statements
   get the variable name on the lhs of the assignment
   search `Tr_var[]` array to check if the variable will be traced or not
   if it is a traced variable then add a call to `update_TT()` after the assignment statement
   (if the variable name is not in `Tr_var[]` array it will not be traced)
   Ex: **a**=a+b; →
   ```
   a=a+b;
   update_TT('a', a);
   ```

Your preprocessor must also write `#include "DebugHdr.h"` at the beginning of the code. The header file will contain the codes of the debugger functions and the data structure used.

expanded.c

```
#include "DebugHdr.h"
int main()
{
      int a, b;
      add_TT('a');
      add_TT('b');
      b=0;
      update_TT('b', b);
      a=1;
      update_TT('a', a);
      while(b<10)
      {
            a=a+b;
            update_TT('a', a);
            b=b+1;
            update_TT('b', b);
            display_TT('b');

      }
      display_TT('a');
      return 0;
}
```

DebugHdr.h

```
// Traces table
struct tr {
      char name;
      int value;
}
struct tr Traces[20];
int nTr;
// Debugger functions
add_TT(char var) {
      ...
}
display_TT(char var) {
      ...
}
update_TT(char var, int val) {
      ...
}
```

The debugger will store the traced variables and their values in the `Traces[]` table in the following structure:

```
struct tr {
      char name;         // variable name
      int value;         // last value
}
struct tr Traces[20];    // max 20 variables
int nTr;                 // number of entries in the table
```

`Traces[]` table is global.

Write the following debugger functions:

`add_TT(char var)`
Allocates an entry in the `Traces[]` table and enters variable *name* in the table

`display_TT(char var)`
searches the variable name in the `Traces[]` table
prints its value on the screen in the following format and waits for the user to enter a character in order to continue
Ex: `a: 5`

`update_TT(char var, int val)`
search the variable name in the `Traces[]` table
enter its *value* to the `Traces[]` table

Implementation

Algorithm of the project:

```
// user interface part
call read_to_buffer()      //read from myprog.c file
start curses mode
while (ch!='x')
        if (ch=='d')
                display buffer on the screen
        if (ch=='w')
                move cursor 1 line up
        if (ch=='s')
                move cursor 1 line down
        if (ch=='i')
                move cursor to the last line
                call add_to_buffer() to  add the directive to Buffer[]
endwhile
end curses mode
call write_to_file()       //write to temp file

// preprocessor part
call read_to_buffer()      //read from temp file
for each line in the Buffer[]
      if the line contains @var
            replace it with the call to  add_TT()
      if the line contains @trace
            replace it with the call to  display_TT()
      if the line contains an assignment statement
            add a call to  update_TT() to update the value of the lhs variable if it is to be traced
endfor
call write_to_file()       //write to expanded.c file
```

NCURSES Library

In the user interface, in order to move the cursor, you need to use NCURSES library. NCURSES can be obtained from ftp://ftp.gnu.org/pub/gnu/ncurses/ncurses.tar.gz or any of the ftp sites mentioned in http://www.gnu.org/order/ftp.html. Read the README and INSTALL files for details on to how to install it. You can find a tutorial in this link: https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/

Example:

```
#include <ncurses.h>
#include <stdio.h>

int main()
{
        int ch;

        initscr();                      /* Start curses mode            */
        printw("Hello World !!!");      /* Print Hello World            */
        refresh();                      /* Print it on to the real screen */
        ch=getch();                     /* Wait for user input */
        //check the character here and process the interface command
        if (ch=='w')
                ...
        if (ch=='s')
                ...
        endwin();                       /* End curses mode              */
        return 0;
}
```

Some functions that you need to use:

`move(row,col)` move the cursor to row*th* row and col*th* column
`addch(ch)` put a single character into the current cursor location and advance the position of the cursor
`getmaxyx(stdscr,row,col)` get the number of rows and columns
`mvprintw(row/2,(col-strlen(mesg))/2,"%s",mesg)` print the message at the center of the screen
`getstr(str)` get a string
`getyx()` can be used to find out the present cursor co-ordinates