Yeditepe University
Department of Computer Engineering

CSE 232
Systems Programming
Fall 2019

Term Project

Debugger/Preprocessor

Due to: 15th December 2019

3 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.

Debugger

Debugger commands:

`@set:` defines which variable will be traced
`@breakpoint:` suspends the execution and allows the user to see the execution trace of a variable

Ex:
If the user wishes to trace variables `a` and `b` at each iteration of the loop in the following code, he/she defines them at the beginning of the code using `@set` and then places breakpoints using `@breakpoint`.

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

Your debugger will have 3 functions:

- **`setVar()`**: used to define which variable will be traced.
- **`bpTrace()`**: suspends the execution and allows the user to see the execution trace of a variable.
- **`saveVar()`**: saves the current value of a variable.

In order to trace a variable, every change in the value of that variable must be saved by the debugger. This will be done by calling `saveVar()` function. Calls to `saveVar()` function must be inserted in the code automatically.

Preprocessor

Write a preprocessor that reads the source code line by line and replaces **`@set`** commands by calls to **`setVar()`** function, **`@breakpoint`** commands by calls to **`bpTrace()`** function.

The preprocessor must also insert calls to `saveVar()` function. If a traced variable is used on the left side of an assignment statement (such as `a= ...`), the preprocessor must insert a call to `saveVar()` function on the next line after the statement, with the name of the variable as its parameter.

Ex:

```
#include "myDebugHdr.h"
int main()
{
	int a, b;
	setVar("a", &a);
	setVar("b", &b);
	b=0;
	saveVar("b");
	a=1;
	saveVar("a");
	while(b<100)
	{
		a=a+b;
		saveVar("a");
		b=b+1;
		saveVar("b");
		bpTrace("b",1);
	}
	bpTrace("a",10);
	return 0;
}
```

Compile your preprocessor code and save its executable file with the name `myPreproc`.

Implementation:

The debugger must save the variable names and their values in array `Traces[]`. The user can trace at most 5 variables. Variable names are maximum 15 characters and for each variable last 20 values can be displayed. Use the following data structure:

```
struct tr {
	char name[15];	// variable name
	int *adr;	// address of the variable
	int values[20];	// last 20 values
	int first;	// points to the index of the first value in values[]
	int last;	// points to the index of the last value in values[]
}
struct tr Traces[5];	// 5 variables
```

Write this data structure in the header file named `myDebugHdr.h`. The user includes it as a header file using: `#include "myDebugHdr.h"`

`Traces[]` must keep the last 20 values of a variable. During the execution of the user program, a variable's value may change more than 20 times. Therefore, the values of a variable in `Traces[]` array need to be managed as a circular list. For this purpose, use `first` and `last` to mark the most recent and the least recent values entered in `values[]`.

Write the debugger functions in the following format:

```
void setVar(char *name, int *adr)
{
	// enters variable name and its address in Traces[]
}
void saveVar(char *name)
{
	// searches variable name in Traces[] and enters its current value in values[]
}
void bpTrace(char *c, int n)
{
	// searches variable name in Traces[] and displays its last n values
	// waits for the user to press Enter
}
```

Write these functions in a file named `myDebugFunc.c` . Also write their prototypes in `myDebugHdr.h`.

Write a **shell script** named `mydebugger` that takes the name of the user program as the command line argument. Your shell script should perform the following:

- Run your preprocessor `myPreproc` to produce the expanded source code named `expanded.c` (where calls to `saveVar()` function are inserted).

- Compile `expanded.c` together with the debugger functions (`myDebugFunc.c)` and run it.

Therefore, if a user wishes to use your debugger for his/her program named `prog.c`, the command

```
./mydebugger  prog.c
```

will start its execution in debug mode.

The structure of `mydebugger` is given below.