

CEng 242 Homework 2

Due: 3rd April 2005

In this homework you will experiment variable lifetimes and write a program to trace lifetimes of global, local and heap variables. You will be using the following type definitions:

```
type Decl = [String]
```

```
data Command = UseVar String | CallFunc String | HeapAlloc String | HeapFree String
              deriving Show
```

```
data Function = Func { name::String, locals::Decl, body::[Command] }
                 deriving Show
```

`Decl` type is a list of strings denoting a group of variable declarations. For this homework, types of variables are not significant so we only mention the variable names.

A `Command` denotes an executable commands accessing variables. Values of this type mean have a reference to variable, call another function, allocate a heap variable, deallocate a heap variable respectively (`UseVar`, `CallFunc`, `HeapAlloc`, `HeapFree`).

A `Function` is a triple denoting a function declaration. First element is the name of the function. Second is the local variables declared in the function and the last one is a list of `Command`'s denoting the executable commands in the function. '{ ... }' notation is a special syntax where the functions `name`, `locals`, `body` that maps `Function` values to their fields are automatically generated by the interpreter. You can use two notations equivalently:

```
Func "main" ["x","y"] [UseVar "x", CallFunc "f"] , or
Func{name="main",locals=["x","y"],body=[UseVar "x", CallFunc "f"]}
```

Given these definitions, you can define a program as a group of global variable declarations and a list of function declarations like:

```
["x","y"] [Func{name="f",locals=["a"],body=[UseVar "x"]},
           Func{name="main",locals=["x","y"],body=[UseVar "x", CallFunc "f"]}]
```

The program above corresponds to the following C program:

```
int x,y;
void f() {
    int a;
    x++;
}
int main() {
    int x,y;
    x++;
    f();
}
```

You will write an Haskell function `trace` with the following signature:

```
trace:: Decl -> [Function] -> IO ()
```

This function will take a global variable declaration list, and then a list of function declarations, and trace this program for lifetimes of variables. If `main` function is given in the function list, it is the first function to call. Depending on the `CallFunc` commands executed, the other functions are called.

The result of the trace is output on the standart output using `putStr`, `putStrLn` calls. Each action in the command trace produces the following output:

Create Variable *var*

This output is generated when a variable is created. These include global variables local variables and heap variables. Global variables created in the declaration order when the program trace starts. Local variables created on entrance to a called function in the order given in `locals` declaration list. A heap variable is created when `HeapAlloc var` is executed.

Destroy Variable *var*

This output is generated when a variable is destroyed. These include global, local and heap variables. Global variables are destroyed in the declaration order when the trace is terminated as the last group of output. Local variables are destroyed in the order of declaration when the trace of the function is ended. A heap variable is destroyed when `HeapFree var` is executed.

Call *func*

This output is generated when a function is called by a `CallFunc func` command.

Return *func*

This output is generated when a function is ended. It is output after the local variables are destroyed.

Invalid reference *var*

This output is generated when an invalid reference to a non-existing (not alive) variable. This might be due to a `UseVar var` or `HeapFree var` command.

Error: main is not found

This output is generated when there is no `main` function is declared in the function declaration list given to `trace` function. Trace simply outputs this and ends.

Function *func* is not found

This output is generated when a function required by `CallFunc func` call is not found.

We have some assumptions to simplify the implementation:

- No recursive calls allowed.
- There is no repeated declarations in the same declaration list (in global and local). No function declaration will include local declaration like: `["x","a","x"]`. Variable names can be re-used in different blocks (i.e. global "x" can be hidden by a local "x" of "main" and it can be hidden by local "x" of "f". It makes three copies of same variable name however in different blocks)
- Heap variables are unique. No local or global variable names are similar with heap variable names. No heap variable will be recreated.
- `HeapFree` will not try to deallocate a global or local variable. It will only be called for variables created by `HeapAlloc`

- Heap variables do not need any carrying of references like pointers, return values etc. They are all created in a global scope and destroyed from a global scope. They are all accessible from anywhere on the program.

Write the `trace` function in a file called `hw2.hs`. You can use standard Haskell library functions. Write any helper functions. Your final submission should include only data definitions and related functions of your implementation. A simple test data will be given in newsgroup soon.

We like to remind you that our cheating policy is to give 0 to all participants for all 6 previous and following homeworks.