Task 2: Escape Detection

Chew Yung Chung	Tan Qiu Hao, Joel	Juliana Seng
a0133662J	a0125473H	a0126332R
a0133662@u.nus.edu	a0125473@u.nus.edu	a0126332@u.nus.edu

Implementation and Algorithm

Within each function, the algorithm maintains a list of new pointers allocated, and when a return instruction is encounter, we check if the variable to be returned matches any entry in the list of newly allocated pointers. If a match is found, an antipattern is detected.

For each function which is not the main() function, a check is performed. The code is as follows:

```
80  ReturnInst *R = dyn_cast<ReturnInst>(&I);
81  if (R) {
82          Value *v = R->getReturnValue();
83          if (v && allocaMap.size() > 0) {

84          pair<multimap<Value*,Value*>::iterator,multimap<Value*,Value*>::iterator> ret;
85                  Value *nextV = 0, *prevV = dyn_cast<Value>(R), *first;
86                  while (true) {
87                          ret = allocaMap.equal_range(v);
88                          for (multimap<Value*,Value*>::iterator it = ret.first; it != ret.second;
    ++it) {
89                                  if (isa<AllocaInst>(v)) {
90                                          if (lineMap[it->second] < lineMap[prevV]) {
91                                                  nextV = it->second;
92                                          }
93                                  } else {
94                                          if (lineMap[it->second] <= lineMap[v]) {
95                                                  nextV = it->second;
96                                                  prevV = v;
97                                          }
98                                  }
99                          }
100                         if (nextV == 0 || v == nextV) {
101                                 if (v && v->getName() != "") {
102                                         errs() << v->getName() << " escaped \n";
103                                 }
104                                 break;
105                         }
106                         v = nextV;
107                 }
108         }
109 }
```

Important Code snippet from asg2-task2.cpp

For each instruction in the function, we build a relationship graph between all store, load, getElementPointer, and cast instructions.

When the algorithm reaches a return instruction, the return value is retrieved for processing. If the return value is not null, the algorithm will attempt to retrieve the variable name that is associated with the return value and compare it against the entries in **`allocaMap`**. If a match is found, the program concludes that the return value is a local pointer, and reports an escape violation.

Build and Run

Run build.sh to compile the test cases, program and execute the test cases.

```
$ ./build.sh
```

The individual commands can also be found in build.sh.

An expected output of the shell code is as follows, all warnings are truncated.

```
----------COMPILING TESTCASES----------


----------COMPILING PROGRAM----------


----------RUNNING TEST 1----------
WARNING: pointer <p> in the function <init_array> will not exist after the return


----------RUNNING TEST 2----------
WARNING: pointer <p> in the function <escape_local> will not exist after the return
```

Test Cases

Test case 1: DCL30-C Antipattern 1

```
$ ./asg2-task2 escape.ll
```

Test case 1 observes the behaviour of escape.c, whereby the return value is a pointer to a locally created array.

```
 7  char *init_array(void) {
 8          char array[10];
 9          char *p = array;
10          return p;
```

```
11 }
```

Code snippet from escape.c

Test case 2: DCL30-C Antipattern 2

```
$ ./asg2-task2 escape2.ll
```

Test case 2 observes the behaviour of escape2.c, whereby the return value is a pointer to a locally created character.

```
7  char *escape_local() {
8          char local_char = 'a';
9          char *p = &local_char;
10         return p;
11 }
```

Code snippet from escape2.c

Corner Cases

Pointers to global variable

```
int globalArr[10];

int* escape_func();

int main () {
        return 0;
}

int* escape_func() {
        int a[10];
        int *p = &globalArr;
        return p;
}
```

In the function escape_func(), the pointer p points to an global array, hence an escape violation does not happen, and the statement is valid. Our algorithm will not detect this as a escape.