Programming Language Design
Assignment 2 2024

Torben Mogensen and Hans Hüttel

February 19, 2024

This assignment is a group assignmentwith groups of 1 to 3 people. Do not seek help from other groups, nor provide them with help.. This will be considered plagiarism. If you use material from the Internet or books, cite the sources. Plagiarism *will* be reported. If you use ChatGPT or similar large language models, you should include the entire dialogue with the AI in your report (otherwise, it is plagiarism), and you should discuss the relevance and correctness of the answers you get (otherwise, you will get no points whatsoever).

Assignment 2 is the second of three mandatory assignments. You are required to get at least 40% of the possible score for every mandatory assignment, so you can not count on passing by the later assignments only. If you know what you are doing, you can solve the assignment in an afternoon, but if you have not or actively participated in the plenary and classroom activities, you should expect to use *considerably* more time.

We advise you not to split the questions between the members of your group. You will get much more benefit if you work on all questions together, and this will prepare you better for the exam assignment.

The deadline for the assignment is **Friday March 1 at 16:00 (4:00 PM)**. The exercises below are given percentages that provide a rough idea how much they count (and how much time you are expected to use on them).

In normal circumstances, feedback will be given by your TA no later than March 8. Note that, even if the TAs find no errors in your submission, this is no guarantee that it is perfect, so we strongly recommend that you take time to improve your submission for resubmission regardless of the feedback you get. You can do so until **March 18 at 16:00 (4:00 PM)**. Note that resubmission is made as a separate mandatory assignment on Absalon.

The assignments consists of several exercises, some from the notes and some specified in the text below. You should hand in a single PDF file with your answers and a zip-file with your code. Hand-in is through Absalon. Your submission must be written in English.

A2.1) (5%) **Stack Allocation (Memory management)**

In Section 4.3 in *Programming Language Design and Implementation*, it is stated that a limitation of stack allocation is that a stack-allocated array can not be returned from the function in which it is declared, as the frame in which it is allocated is freed at function return. There are, however, methods for getting around this restriction.

a. Describe a method for returning an array from the function in which it is declared. Assume that arrays are stack allocated. Describe limitations (if any) to your method.

A2.2) (30%) **Garbage Collection (Memory management)**

This question uses the conservative mark-sweep collector presented in the session about memory management.

The isHeapPointer() function assumes that pointers into the heap always point to the first field of an object, just after the header (which consists of the magic word, the object size and a mark bit). In C, it is possible for a pointer to point to any field of an object and to the address immediately after the last field. Anything else is deemed undefined behaviour.

The mark() function assumes that pointers are to the first field of an object (which the isHeapPointer() function verifies), but this is no longer the case. When adding a pointer to the stack, it should be a pointer to the first field of an object, and not a pointer into any other part of the object.

a. Modify the isHeapPointer() function to allow arguments that are pointers into objects as described above. If the argument is not such a pointer, isHeapPointer() should return 0. If it is (or, rather, could be) such a pointer, isHeapPointer() should return a pointer to the first field of the object. Note that this changes the type of the function, which should now be

```
uint64_t *isHeapPointer(uint64_t *value)
```

b. Modify the mark() function to handle pointers into objects as described above. **Hints:** You should only push pointers to the first fields of objects to the stack. You can exploit that, if argument is a pointer into a heap object, the new isHeapPointer() function returns a pointer to the first field of that object.

c. To test the modified functions, we modify the main() function to

```
int main() {
  uint64_t *ll = NULL;
  uint64_t *cc = NULL;

  initialize_freelist();

  for (int i = 0; i<60; i++) {
    if ((cc = allocate((3*i)%11+3)) != 0) {
      printf("allocation successful: %ld\n", cc - heapStart);
      cc[0] = 0x1001*i;
      if (i%13 == 0) cc[1] =  firstGlobal[i%17]; else cc[1] = 0;
      ll = cc;
      firstGlobal[i%17] = (uint64_t) (ll+i%3);
    } else {
      printf("Allocation failed at i == %d, size = %d\n", i, (3*i)%11+2);
      exit(0);
    }
  }
}
```

which you can find in the file `newmain.c`.

Run the modified program and show the lines of output containing the word "sweep" (on Linux or MacOS, you can run gc1 | grep sweep, where gc1 is the name of the modified gc program). This will indicate the number of garbage collections done before allocation fails.

A2.3) (15%) **(Scope rules)** Give outlines of programs in a language with static scope rules in which

a. two syntactically identical function definitions (in the same program) define functions that are not equivalent

b. a single occurrence of a function definition is interpreted more than once and the two interpretations define functions that are not equivalent.

In both cases, also give an example of identical calls (in different scopes) that would give different answers.

A2.4) (20%) **(Parameters)** In some programming languages, every programmer-defined function has a fixed number of parameters. However, some of these languages also have examples of existing functions that have variable-length lists of actual parameters.

- Give examples of predefined functions of this kind.
- Discuss the advantages and disadvantages of having variable-length parameter lists.
- Suggest an extension to the syntax of your favourite imperative programming language that does not have this feature but would allow programmer-defined functions to have variable-length parameter lists. Argue why this is a good way to design such an extension.

A2.5) (30%) **(Program transformations)**

Here is a Haskell program `prog1.hs`, which you can find in the file of the same name.

```
bizarre g x = let m z = g z in
                m x

mango f u v = f v u

dingo w =  let plip k m = w k m in
             let plop n = plip n 484000 in
                 plop 17

plys x y = x + 4

ninka x = 0 - x

zap x y = x*y

mina = mango plys (bizarre ninka) (dingo zap)
```

(a) Use lambda-lifting on prog1.hs to produce an equivalent program prog2.hs. Explain what lambda lifting achieves.

(b) Show the result of evaluating mina using GHCi and explain the result.

(c) Use defunctionalization on prog2.hs. Explain what defunctionalization achieves.