University of Oxford
Department of Computer Science

Compilers

Michaelmas Term 2020

Your report on this assignment should be submitted online via WebLearn by 12 noon on 29th January 2021. The assignment is worth 35 marks out of a total of 100 marks for the course.

This assignment is based on the compiler from a Pascal subset to code for the ARM that was the subject of Lab 4 in the course. It asks you to make two extensions, which are specified later in this document.

To carry out the assignment, you will need access to a Unix environment containing version control tools, an OCaml compiler, software tools that enable cross-compiling for the ARM, an ARM emulator, and other support software. Hints and resources for setting up such an environment are provided on the course web page. Lab machines are available for remote access.

A fresh copy of the practical materials for the course may be obtained as described on the course website, e.g. by cloning the Git repository supplied for the lab exercises to a directory named `task`:

```
$ git clone http://spivey.oriel.ox.ac.uk/git/compilers task
```

Candidates are required to submit a written report, describing the changes they made to the compiler in order to support the extensions, and the provisions they have made for testing the modified compiler.

A typical report will consist of five to ten pages of text with embedded code fragments, plus listings of the compiler changes and additional test cases. A sample report for a different exercise is provided on the course web page, and shows the suggested structure, with a narrative account of the compiler changes stage by stage, followed by difference listings generated with a version control system, and test cases in the format of the existing tests for Lab 4, showing the compiler input, the expected output, and the assembly language code. Candidates are not required or permitted to submit the code of their implementation in machine-readable form.

Your attention is drawn to the University's policy on Plagiarism set out in Appendix A of each Course Handbook and on the University's website. The work submitted for this assignment should be entirely your own and not done in collusion with others. You may make use of written and online sources, but these should be acknowledged.

The remainder of the document sets out the two tasks that constitute the assignment.

Task 1: Breaking out of nested loops

The first task concerns the implementation of a `break` n statement, which modifies the behaviour of loops. Note that the language of Lab 4 features three loop statements (`for`, `repeat` and `while`). Each of them counts as a loop for the purpose of the assignment.

- `break` accepts an argument n, which tells it how many enclosing loops should be broken out of.
- The argument must be a positive integer constant and the compiler should report an error if this condition is violated. The argument is optional, on the understanding that `break` stands for `break` 1.
- Each occurrence of `break` n should be surrounded by at least n loops. If this is not the case, the compiler should report an error.
- On reaching `break` n, program control should be transferred immediately to the program point following the end of the nth enclosing loop.

The goal of the task is to add `break` n to the language as a new kind of *statement* and extend the compiler so that it conforms to the specification given above.

Example

```
var a: array 10 of array 10 of integer;
var i, j: integer;

begin
  for i:=0 to 10 do for j:=0 to 10 do a[i][j]:=i*j end end;

  i:=0;
  while i<10 do
    j:=0;
    repeat
      if a[i][j]=63 then break 2 else j:=j+1 end
    until j=10;
    i:=i+1
  end;

  (* i=7 and j=9 *)

  print_num(i); newline(); print_num(j); newline()
end.
```

Replacing `break 2` with `break` (or, equivalently, with `break 1`) should change the final values of i and j to 10 and 7 respectively.

Task 2: Memory alignment

The Lab 4 compiler generates code that adheres to a set of memory alignment policies, such as the requirement that integer values be stored at addresses divisible by four. Due to padding, this often results in allocation of memory that will never be used.

In this task, you are asked to design and implement measures that will attempt to reduce the amount of padding and, thus, improve the memory footprint of programs while still observing all alignment constraints.

Your optimisations need not result in optimal or padding-free memory allocation in all cases, but rather handle a range of interesting cases in which optimisation is possible. For example, the current compiler will use four words of memory to accommodate records of type `record f1:boolean; f2:integer; f3:boolean; f4:integer end`. However, if the fields were stored in a different order, three words would suffice to hold the content without violating the alignment rule for integers.

Please make sure to explain in the report what scenarios are and are not covered by your method. You may wish to start off by reviewing the compiler code and identifying all places affected by alignment conventions.

Assessment

Partial marks will be given for implementing only some aspects of the assignment, or generating code that is significantly inefficient or over-long. To gain full marks, your submission should include test cases that illustrate the quality of your implementation and the generated code. The following mark scheme will be used, giving a maximum total of 35 marks.

- A basic implementation of Task 1 with simple test cases showing the code.

  (10 marks)

- A basic implementation of Task 2 with simple test cases showing the code.

  (10 marks)

- Tidy object code of reasonable efficiency, demonstrated by test cases.

  (5 marks)

- Further test cases that demonstrate subtle aspects of the solution.

  (5 marks)

- A written report with good clarity and presentation.

  (5 marks)