Formal languages and compilers
Projects for the exam

Academic year 2010-2011

General information

The project consists of the extension of the interpreter or compiler[1] presented during the lecture. The project is provided for groups of no more than three persons, and should be chosen from the options offered below, or agreed with the teaching assistant *before* starting.

For the project to be evaluated it is crucial to:

- present the code of the interpreter/compiler extended according to the option of the project
- submit a report on the project according to the description in Section 4.
- the compilation must not return errors. Not serious warning can be accepted if properly justified in the report. Example: "the warning *this pattern-matching is not exhaustive* in line xyz it is due to the fact that I guarantee that the type cannot be different from those indicated in the match because..., and so I decided not to consider all possible cases."
- the working examples in extended version of `créme CAraMeL` should reflect the intention of the chosen project.

1 Groups of 1 person

A free choice of the project from the following list.

1.1 Pointers and dynamic memory management

Introduce in `créme CAraMeL` the possibility to use pointers and dynamic memory (heap) as follows:

Pointers: add to the language the possibility to use pointers as in the following examples:

[1]It's a free choice, but take into account that the working project based on compiler will be given a bigger value with respect to the same (working) project based on the interpreter.

- declaration of the pointer variables:

  ```
  var p : ^int;
  var q: ^^float;
  ```

- referencing and dereferencing pointers:

  ```
  x := 1;
  p := @x;
  y := ^p + 4;
  ```

  the result will be `y = 5`.

Notice that pointers can have *an arbitrary depth* (number of " ^ " in the declaration).

Dynamic memory: based on what was discussed at the lecture, introduce in the language the dynamic memory management as *heap*: a memory separated from the one currently implemented (that is represented with the *stack* where local variables of programs and subprograms are allocated) in which it is possible to allocate objects at run-time. The deallocation of dynamic memory must be based on one of the following approaches:

- reference counter: add counters to the "cells" of the memory so that there will be a way to count how many pointers point to each cell, and remove any cells that appear to be no longer used.
- garbage collection: use the algorithm of mark/sweep presented in class to perform the actual release of unused memory cells.

1.2 Pointers and record

Introduce in `créme CAraMeL` the possibility to use pointers and record as follows:

Pointers: see Section 1.1 (only the part on pointers)

Record: Extend the language to give a possibility to define and use the record type, by making the necessary modifications so that *all* of the following situations are supported:

- definition of `record` type

  ```
  type name_record = record {
  name_field_1 : type ;
  ...
  name_field_n : type ;
  }
  ```

- declaration of variables of type `record`:
  `var v :` *`name_record`*`;`

- mechanisms to access the components of type `record`:
  `v.`*`name_field`*$_i$ `:=` *`expression`*`;`
  `a := v.`*`name_field`*$_i$ `;`

- the possibility to compare $(<, \leq, =)$ the objects of type `record`: given that `v` and `w` are of the same record type containing two fields `a: int` and `b: int`, it should be the case that:
  ```
  v.a := 4; v.b := 6;
  w.a := 3; w.b := 1;
  if (w < v) then write(1) else write (0);
  ```
  will write `1` on the screen.

- the assigning command between the objects of type `record`: with the same assumptions as above, an assignment:
  `v := w;`
  have to give a result `v.a = w.a = 3` and `v.b = w.b = 1`.

1.3 Array implemented by linked lists

Based on the examples seen at the lecture, change the implementation of `créme CAraMeL` in a way that the operations that need changing the dimension of the vector, such as inserting or deleting elements, are possible. It is suggested to implement vectors using the linked lists and take into account that the following operations are possible:

- inserting an element
  `v(3) := 5`
  this command inserts in vector `v` a value 5 on the third position, increasing the length of the vector.

- substitution of an element
  `v[2] := 6`
  this command substitutes the second element in vector `v` with a value 6.

- deleting an element
  `v#4`
  this command deletes the forth element from the vector `v` a value 5, decreasing the length of the vector.

- finding a value:
  `v?23`
  this expression has a value `i` if vector contains value 23 at position `i` and has a value `-1` if there is no value in the vector that is equal to 23.

2 Groups of 2 persons

A free choice of the project from the following list.

2.1 Multidimensional matrices and slices

Introduce in `créme CAraMeL` the possibility to use multidimensional matrices *and* slices:

Multidimensional matrices

Introduce in `créme CAraMeL` the possibility to use multidimensional matrices (= array with n dimensions) as it was described in the class.

- It should be possible to declare in the following ways
  `var v : array [`LB_1`..`UB_1`, `LB_2`..`UB_2`, ..., `LB_n`..`UB_n`] of `*type*`;`
  `var v : array [`LB_1`..`UB_1`] of`
  `array [`LB_2`..`UB_2`] of ... array[`LB_n`..`UB_n`] of `*type*`;`
  as well as in the combination of them:
  `var v : array [`LB_1`..`UB_1`] of`
  `array [`LB_2`..`UB_2`, ..., `LB_n`..`UB_n`] of `*type*`;`

- The syntax for accessing the elements can be done (by free choice) in one of the following ways:
  `v[i][j] := 5; write(v[i][j] + 2);`
  `v[i, j] := 5; write(v[i, j] + 2);`

Slices

Based on the examples seen at the lecture, introduce in `créme CAraMeL` the possibility to use the slices as follows:

- declaration:
  `var v : array [`LB_1`..`UB_1`, `LB_2`..`UB_2`, ..., `LB_n`..`UB_n`] of `*type*`;`
  `var s: slice [*, *, ..., j, *, k, *, ..., *] of v;`
  two indexes `j` and `k` of the array `v` are fixed in the slice `s` (but one can fix an arbitrary number of indexes). Example:
  `var v : array [5..43, 4..17] of int;`
  `var w : slice [*, 10] of v;`
  `w` is the slice that takes only column number 10 of the matrix `v`. Hence, the indices of `w` are from 5 to 43.

- accessing the elements:
  considering the example above, the following accesses should be allowed:
  `w[6] := 4 + 12;`
  `write(w[40]);`
  but this is not a correct access

```
w[3] := 2;
```
because 3 is not a valid index ($3 \notin \{5, ..., 43\}$)

2.2 Pointers and different ways of passing the parameters

Add to the language `créme CAraMeL` *all* of the following functionality:

Pointers : see a part ”Pointers” in Section 1.1

Passing the parameters : Introduce in `créme CAraMeL` the possibility to pass the parameters to the procedure and functions in *all* the possible ways:

- value
- reference
- value-result
- name

according to the way shown at the lecture.

3 Groups of 3 persons

A free choice of the project from the following list.

3.1 Record, pointers, multidimensional matrices and slices

Implement *all* of the following funcitonality:

- multidimensional matrices and slices (see Section 2.1)
- pointers (see a part ”Pointers” in Section 1.1)
- record (see a part ”Record” in Section 1.2)

3.2 Functions and nested procedures, with different ways of passing parameters

Add to the language `créme CAraMeL` *all* of the following functionality:

Functions and nested procedures : add the possibility to define functions and nested procedures (inside the other functions/procedures). Define the environment of the procedures/functions using the rule of static scoping based on the scoping tree as it was shown at the lecture.

Passing the parameters : see the Section 2.2 (implementation of the pointers is not mandatory)

4 Delivery

What to deliver

- report **written in English** of about **10-15 pages** explaining the details of the project:
  - objective
  - the work done (what are the modifications that were done and why)
  - principal choices of implementation
  - difficulties that were faced (the choices of implementation that failed and why, a way of solving the problems, ...)
  - names and matriculation numbers of the participants of the project
- code of the interpreter/compiler extended according to the tasks of the chosen project
- code in `créme CAraMeL` of several examples showing the functionality added to the language, commented to explain the meaning and the output they should produce. All the comments should be in English.

When to deliver strictly before and not later than *12:00* of the day indicated on the website of the course `http://disi.unitn.it/~bielova/flc/index.html`

How to deliver via e-mail to the address `bielova@disi.unitn.it`, attaching the report (in format .pdf) and the source code + examples (compressed in format .zip or .tar.gz).

The oral evaluation of the projects will be done during the exam.