Subprograms in "crème CAraMeL"

Lecture 9

Formal Languages and Compilers 2011

Nataliia Bielova

Assumptions and simplifications

- Let's add
  - declarations of subprograms (procedure)
  - execution of subprograms (call and passing the parameters)
- no declarations inside the blocks `begin … end`
- declarations are non-static: dynamic local environment
- only one environment, no activation record: dynamic non local environment
- passing the parameters only *by value*

Procedure: example

```
program
  var x : int

  procedure proc1(a: int)
  begin
    write(a)
  end;

  procedure proc()
    var x : int
  begin
    x:=5;
    call proc1(x);
    write(x)
  end

begin
  x := 40;
  write(x);
  call proc();
  call proc1(x)
end
```

⇒

```
40
5
5
40
```

Procedure: implementation

Syntax:

- parser.mly: new token PROCEDURE, CALL, COMMA (“,”)
- lexer.mll: strings corresponding to token
- syntaxtree.ml: constructors for
  - declarations
  - formal parameters
  - calls
  - actual parameters
  - other modifications
- parser.mly: productions to construct new nodes

Procedure: implementation

Semantics:

- new value in the environment:

```
Descr_Procedure of param list * dec list * cmd
```

- declaration
- execution (call)
  - evaluation of actual parameters
  - type checking for the list of parameters
  - actual execution of the procedure

Function: example

```
program
  var x : int

  function fact(a: int): int
    var b : int
  begin
    if (a = 0) then
      fact := 1
    else begin
      b := call fact(a - 1);
      fact := a * b
     end
  end

begin
  x := call fact(12);
  write(x)
end
```

⇒

```
479001600
```

Function: implementation

- Syntax:
  - keyword: function
  - new nodes for: declaration, execution (call), evaluation (call!)
  - adjust the syntax tree
- Semantics:
  - declaration (attention: a location for return value is needed!)
  - evaluation
  - execution

Projects for the exam

Extension of an interpreter/compiler:

- 1 person (either one or another item)
  - pointers and dynamic memory
  - pointers and record
  - vectors "by linked list"
- 2 persons (also here)
  - multidimensional matrices and horizontal-vertical slices
  - pointers, passing parameters by value, name, reference, value-result
- 3 persons (like before)
  - record, pointers, multidimensional matrices and horizontal-vertical slices
  - definition of functions and nested procedures (static scoping) and passing parameters by value, name, reference, value-reference

Pointers

- Declaration:

```
var p : ^int;

var q: ^^float;
```

- Referencing(@) and dereferencing (^) :

```
x := 1;

p := @x;

y := ^p + 4;
```

If `x` and `y` are integers, then in the end `y = 5`.

Dynamic memory

Add to the language the possibility to use pointers and allocation/deallocation of dynamic memory (heap), using *one* of the following approaches of memory release:

- reference counter
- garbage collection

A correct implementation will allow to create and use the dynamic data structures using pointers in crème CAraMeL.

Vectors “by linked list”

Change the implementation of vectors in crème CAraMeL in a way that the following operations are possible:

`v(i) := 5` inserting an element (growing the length of the vector)

`v[i] := 5` substitution of an element (the length remains the same)

`v#i` deleting an element (the vector becomes shorter)

`v?5` returns an integer `i` if vector contains value `5` at position `i` and an integer `-1` if there is no value `5` in the vector

Record

- Definition:

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

- Declaration:

```
var v : name_record;
```

- Access:

```
v.name_field_i := expression;
a := v.name_field_i;
```