

## Exercise sheet 6 for 19 March 2003

2003-03-12

Do exercises 6.1, 6.2, 6.3. Hand in the solutions.

**Exercise 6.1** Write some micro-C programs to understand the use of arrays, pointer arithmetics, and parameter passing. Use the micro-C implementation in `imp/c.sml` and the associated lexer and parser to run your programs. Be careful: there is no typechecker and nothing that prevents you from overwriting arbitrary store locations by mistake, causing your program to produce unexpected results. (The type system of real C would catch some of those mistakes at compile time).

For instance:

- Define a function `void arrsum(int n, int a[], int *res)` that computes and returns the sum of the elements `a[0]` through `a[n-1]` of an array.
- Define a function `void squares(int n, int a[])` that, given `n` and an array `a` of length `n` (or more) fills `a[i]` with `i*i` for `i = 0, ..., n - 1`.
- Use function `squares` above to fill an array with `n` squares (where `n` is given as a parameter to the main function), then use function `arrsum` above to compute the sum of the array's elements.
- Define a function `void linsearch(int x, int len, int a[], int *res)` that searches for `x` in `a[0..len-1]`. It should return the least `i` for which `a[i] == x` if one exists, and should return `-1` if not `a[i]` equals `x`.
- Define a function `void binsearch(int x, int n, int a[], int *res)` that searches for `x` in a sorted array `a[0..n-1]` using binary search. It should return the least `i` for which `a[i] == x` if one exists, and should return `-1` if no `a[i]` equals `x`.
- Define a function `void swap(int *x, int *y)` that swaps the values of `*x` and `*y`.
- Define a function `void sort(int n, int a[])` that sorts the array `a[0..n-1]` using selection sort. This function should use function `swap` above.

**Exercise 6.2** Extend the micro-C abstract syntax in `imp/Absyn.sml` with the postincrement and postdecrement operators known from C, C++, Java, and C#:

```
datatype expr =
  ...
  | PostInc of access (* Java i++ or a[e]++ *)
  | PostDec of access (* Java i-- or a[e]-- *)
```

Note that the postdecrement and postincrement operators work on lvalues, that is, variables and array elements, and more generally on any expression that evaluates to a location.

Modify the micro-C interpreter in `imp/c.sml` to handle `PostInc` and `PostDec`.

**Exercise 6.3** Extend the micro-C lexer and parser to accept `e++` and `e--` also.

**Exercise 6.4** Add compound assignments `+=` and `*=` and so on to micro-C. The left-hand side of a compound assignment must be an lvalue as for ordinary assignment, but it is used also as an rvalue.

**Exercise 6.5** Extend the micro-C lexer and parser to accept C/C++/Java/C# style conditional expressions

```
e1 ? e2 : e3
```

with abstract syntax `Cond(e1, e2, e3)`.