FIXME **T-501-FMAL Programming languages, Assignment 4**
Spring 2021
Due Fri 9 April at 23:59

1. Consider the following MicroC code:

```
int *f(int *t, int *u) {
    if (*t == 0) {
        *t = 50;
        return u;
    }
    else {
        return t;
    }
}

void g(int x, int y) {
    f(&x, &y);
    f(&x, &y);
    print y;
}

void h(int x, int y) {
    int *p;
    p = &x;
    p = f(p, &y);
    f(p, &y);
    print y;
}
```

   What will be printed by the following function calls? In each pair, if the value printed by `g` is different from the value printed by `h` then explain why.

   (i) `g(1, 2)` and `h(1, 2)`

   (ii) `g(1, 0)` and `h(1, 0)`

   (iii) `g(0, 0)` and `h(0, 0)`

The file `Assignment4.fs` contains an implementation of a simplified version of MicroC. The main differences are:

- Expressions do not modify the store.
- Functions do not return values.
- Local variables have to be declared at the beginning of the function. They are initialized to `0`.
- There are no types (local variables are declared in the concrete syntax by writing `var a` instead of e.g. `int a` or `int *a`).
- There are memory allocation statements `a = alloc(e)`, which means allocate space for an array of length `e`, initialize all of the values in the array to to `0`, and set `a` to the address of the first value. In the abstract syntax `Alloc(a, e)` means `a = alloc(e)`. You can assume that allocation of memory will never fail.

`Assignment4.fs` contains some examples of the concrete and abstract syntax.

2. Convert the following three functions to abstract syntax. (Complete the definitions of `print_array`, `memcpy` and `make_copy` in `Assignment4.fs`.)

```
void print_array(a, length) {
  var i;
  while (i < length) {
    print(*(a + i));
    i = i + 1;
  }
}

void memcpy(dest, src, length) {
  while (length) {
    *dest = *src;
    dest = dest + 1;
    src = src + 1;
    length = length - 1;
  }
}

void make_copy(dest_p, src, length) {
  *dest_p = alloc(length);
  memcpy(*dest_p, src, length);
}
```

3. Linked lists of integers can be represented as follows: the empty list `[]` is the integer `0` (serving as a null pointer), the list `x::xs` is represented as a pointer `p` such that `*p` is `x` and `*(p+1)` is `xs`.

   (i) The following is a function, in abstract syntax, that converts an array to a linked list. Render the abstract syntax as concrete syntax.

```
let array_to_list =
  ("array_to_list", ["dest_p"; "a"; "length"], ["cur"], Block [
    Assign (AccDeref (Access (AccVar "dest_p")), Num 0);
    While (Access (AccVar "length"), Block [
      Assign (AccVar "length", Op ("-", Access (AccVar "length"), Num 1));
      Alloc (AccVar "cur", Num 2);
      Assign (AccDeref (Access (AccVar "cur")),
        Access (AccDeref (Op ("+", Access (AccVar "a"),
          Access (AccVar "length")))));
      Assign (AccDeref (Op ("+", Access (AccVar "cur"), Num 1)),
        Access (AccDeref (Access (AccVar "dest_p"))));
      Assign (AccDeref (Access (AccVar "dest_p")), Access (AccVar "cur"))
    ])
  ])
```

   (ii) Write a function `print_list` that takes a linked list as an argument and prints it. Give the function as abstract syntax by completing the definition of `print_list` in `Assignment4.fs`.

4. `TestAndSet(p, q)` means set the value at address `p` to the value at address `q`, then set the value at address `q` to `1`. Implement the `TestAndSet` case of the `exec` function.

5. (i) Explain why the following program prints `10`:

```
void main() {
  var p, q;
  p = alloc(1);
  q = alloc(1);
  *(q - 1) = 10;
  print(*p);
}
```

(ii) Explain why the following program prints `0`:

```
void f() {
  var i;
  while (*(&i + i) != 1234) {
    i = i + 1;
  }
  *(&i + i) = 0;
}

void main() {
  var a, b, c, d;
  a = 1234;
  f();
  print a;
}
```