

# Homework 3

Due 19 October

Handout 3  
CS242: Autumn 2005  
12 October

---

## Reading

---

1. Read chapter 6 on the Algol family and ML, and chapter 7 on scope and storage management.

---

## Problems

---

1. .... ML Types

Explain the ML type for each of the following declarations:

- (a) `fun a(x,y) = x+2*y;`
- (b) `fun b(x,y) = x+y/2.0;`
- (c) `fun c(f) = fn y => f(y);`
- (d) `fun d(f,x) = f(f(x));`
- (e) `fun e(x,y,b) = if b(y) then x else y;`

Since you can simply type these expressions into an ML compiler to determine the type, be sure to write a short *explanation* to show that you understand why the function has the type you give.

2. .... Types and Garbage Collection

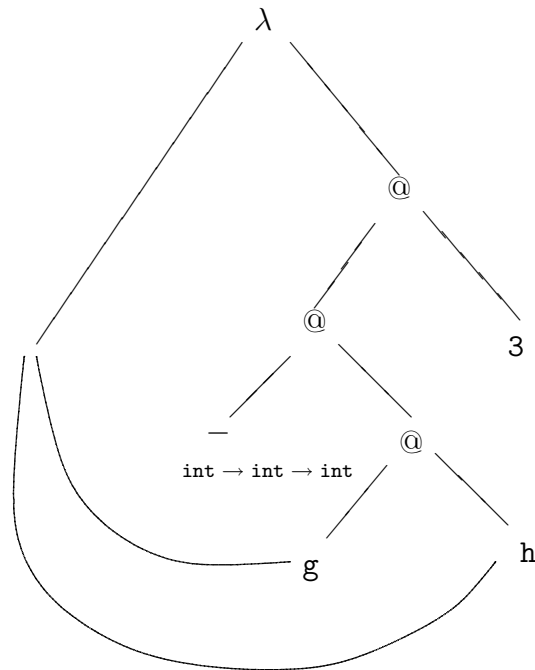
Language  $D$  allows a form of “cast” where an expression of one type can be treated as an expression of any other. For example, if  $x$  is a variable of type integer, then  $(string)x$  is an expression of type string. No conversion is done. Explain how this might affect garbage collection for language  $D$ .

For simplicity, assume that  $D$  is a conventional imperative language with integers, reals (floating-point numbers), pairs and pointers. You do not need to consider other language features.

3. .... Parse Graph

Use the parse graph below to calculate the ML type for the function

```
fun f(g,h) = g(h) - 3;
```



4. .... Parameter Passing

Consider the following procedure, written in an Algol/Pascal-like notation:

```

proc power(x, y, z:int)
begin
  z := 1
  while y > 0 do
    z := z*x
    y := y-1
  end
end

```

The code that makes up the body of `power` is intended to calculate  $x^y$  and place the result in `z`. However, depending on the actual parameters, `power` may not behave correctly for certain combinations of parameter-passing methods. For simplicity, we only consider call-by-value and call-by-reference.

- (a) Assume `a` and `c` are assignable integer variables with distinct L-values. Which parameter-passing methods make  $c = a^a$  *after* a call `power(a, a, c)`. You may assume that the R-values of `a` and `c` are non-negative integers.
- (b) Suppose that `a` and `c` are formal parameters to some procedure `P`, and that the expression `power(a, a, c)` above is evaluated inside the body of `P`. If `a` and `c` are passed to `P` by reference, and become aliases, then what parameter passing method(s) will make  $c = a^a$  *after* a call `power(a, a, c)`? If, after the call,  $c = a^a$ , does that mean that `power` actually performed the correct calculation?

5. .... Function Returns and Memory Management

This question asks about memory management in the evaluation of the following statically-scoped ML expression.

```

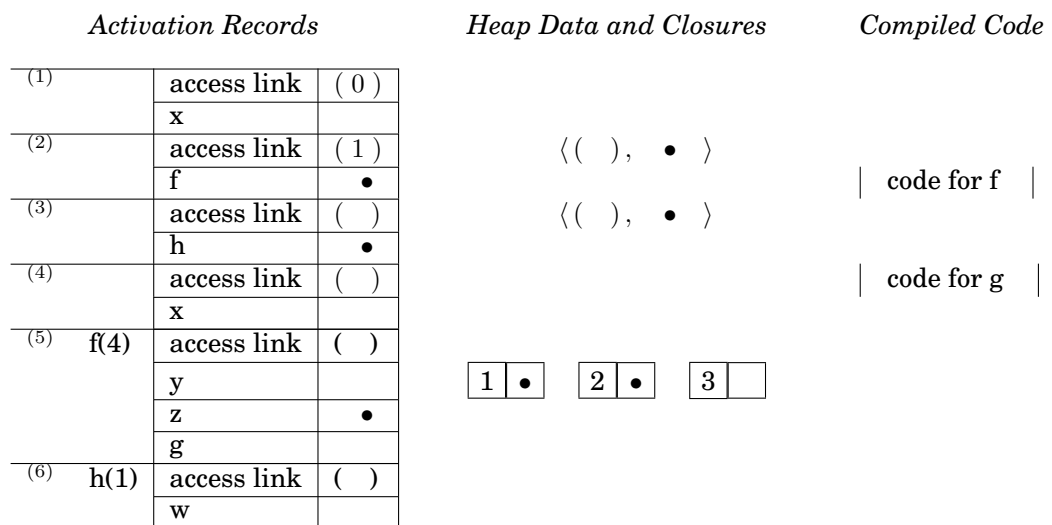
val x = 3;
fun f(y) =
  let val z = [2, 3, 4] (* declare list *)
      fun g(w) = w+x+y (* declare local function *)
      in
        g (* return local function *)
      end;
val h = let val x=6 in f(4) end;
h(1);

```

- (a) Write the type of each of the declared identifiers (*x*, *f*, and *h*).
- (b) Since this code involves a function that returns a function, activation records cannot be deallocated in a LIFO stack-like manner. Instead, let us just assume that activation records will be garbage collected at some point. Under this assumption, the activation record for the call *f* in the expression for *h* will still be available when the call *h*(1) is executed.

Fill in the missing information in the following depiction of the run-time stack after the call to *h* at the end of this code fragment. Remember that function values are represented by closures, and that a closure is a pair consisting of an environment (pointer to an activation record) and compiled code.

In this drawing, a bullet (•) indicates that a pointer should be drawn from this slot to the appropriate closure, compiled code or list cell. Since the pointers to activation records cross and could become difficult to read, each activation record is numbered at the far left. In each activation record, place the number of the activation record of the statically enclosing scope in the slot labeled “access link.” The first two are done for you. Also use activation record numbers for the environment pointer part of each closure pair. Write the values of local variables and function parameters directly in the activation records.



- (c) What is the value of this expression? Explain which numbers are added together and why.
- (d) If there is another call to *h* in this program, then the activation record for this closure cannot be garbage collected. Using on the definition of garbage given in the Lisp chapter, explain why, as long as *h* is reachable, mark-and-sweep will fail to collect some garbage that will never be accessed by the program.

## 6. .... Stack Smashing, Type Checking & Buffer Overruns

A type of attack known as *stack smashing* has become a very popular method for attacking programs written in C and C++. This type of attack is a particular case of the infamous buffer

overflow vulnerability. If you are interested in finding out more, a complete tutorial on this attack can be found at <http://www.insecure.org/stf/smashstack.txt>. We want you to identify some of the consequences of being able to do pointer arithmetic, i.e. to adjust pointers as if they were integers and modify the contents of the new integers.

- (c) Consider this example program (to build it, feel free to run `gcc-2.95 -g file.c` on any of the firebird machines with your SUNet login):

*//courtesy Phrack Volume 7 Issue 49 by Aleph One*

```
int myvar=8;
void donothing(int a, int b, int c) {
    int buffer1;
    int *ret;
    ret = &buffer1+2;    //this increments the pointer by 8 bytes
    (*ret) += myvar;
}

int main() {
    int x=0;
    donothing(1,2,3);
    x = 1;
    printf("%d\n",x);
    return 0;
}
```

Running this program results in 0 as the output. Explain and draw a diagram of how memory could be layed out to cause such a problem.

- (b) What could happen if you set `myvar` to be -8?
- (c) Counting votes. In the light of the new automatic voting machines, it has come to the attention of many how important a paper trail verifyable by the user is. Just a few lines of code can turn an innocuous program which works properly into something far more insidious. Last year before November 2 there was a contest to determine how a very simple program that read in letters and tallied them up could be skewed on certain days of the year. Paul Parkanzky's program was the winner of the 2004 Obfuscated-V contest. It tries to count votes, yet it alters the result on the November 2 only. The technique used is related to those discussed before, it overruns a fixed-length array in the program.

Explain how it gets the wrong answer. (Hint: the `LogMsg` string is only 11 characters long! Also: assume `printResults` and `paperTrail` work properly and adjust `getDay` so it's always Nov 2.)

```
/* http://graphics.stanford.edu/~danielrh/vote/pparkanzky.c
 * On Nov 1 the votes are counted correctly. On Nov 2 the total is
 * correct, but one candidate is "delivered" extra votes at
 * another's expense.
 *
 * setenv PATH=/usr/bin:$PATH
 * gcc-3.3 file.c (on raptor or vine)
 */
#include<stdio.h>
#include<ctype.h>
#include<time.h>

void printResults(unsigned int *);
```

```

void paperTrail(char *);
char *getDay();

int main() {
    unsigned int Tally[4] = {0};
    unsigned char Other;
    unsigned char Nader;
    unsigned char Bush;
    unsigned char Kerry;
    char LogMesg[11] = {0};
    int Input;
    char *day; day = getDay();
    Bush = 0;
    Kerry = 1;
    Nader = 2;
    Other = 3;
    while ((Input=getchar())!=EOF) {
        unsigned char Vote=Input;
        sprintf (LogMesg,"LOG VOTE: November %s %c\n",day,Vote);
        paperTrail(LogMesg);
        if (!isspace(Vote)){
            switch (Vote) {
                case 'B':
                    Tally[Bush]++;    break;
                case 'K':
                    Tally[Kerry]++;    break;
                case 'N':
                    Tally[Nader]++;    break;
                default:
                    Tally[Other]++;    break;
            }
        }
    }
    printResults(Tally);
    return 0;
}

char *getDay() {
    char *retVal;
    time_t now_time;
    struct tm *now;

    now_time = time(NULL);
    now = localtime(&now_time);
    if (now->tm_mday == 1) {
        retVal = "first";
    } else if (now->tm_mday == 2) {
        retVal = "second";
    } else if (now->tm_mday == 3) {
        retVal = "third";
    } else {
        retVal = (char *)malloc(7);
        sprintf(retVal,"%ith",now->tm_mday);
    }
}

```

```
        return retVal;
    }
void printResults(unsigned int *Tally) {
    unsigned char Bush = 0;
    unsigned char Kerry = 1;
    unsigned char Nader = 2;
    unsigned char Other = 3;
    printf("Kerry: %d\n",Tally[Kerry]);
    printf("Bush: %d\n",Tally[Bush]);
    printf("Nader: %d\n",Tally[Nader]);
    printf("Other: %d\n",Tally[Other]);
}
void paperTrail(char *mesg) {
    /* Log to paper trail for extra legitimacy! */
}
```