Compilers

Spring 2009

Homework 5

Due Tuesday, May 26, 2009 at 2.30 PM in class

Please label all pages you turn in with your name and student number.

Problem 1: Call Graph and Call Tree [10 points]

Consider the following C program:

```
#include < stdio.h>
#include <stdlib.h>

int A(int a){
  D(a);
  return B(a-1) + 1;
}

int B(int b){
  if(b <= 0){
     return –b;
  } else {
     return A(b-1) + 1;
  }
}

int C(int c){
  return (c* c);
}

void  D(int d){
  int x;
  printf("at D: %d\n",d);
}

int main(int argc, char ** argv){
   int x, y;

   x = B(atoi(argv[1]);
   y = C(x);
   printf("x, y = %d, %d\n",x,y);
}
```

Questions:

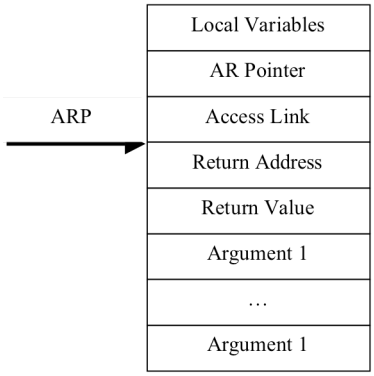
(a) Show its call graph, *i.e.* caller-callee relationship for user defined procedures/functions.
(b) Show its call tree and its execution history, *i.e.,* the arguments' values and output produced when the value of argv[1] is '2.
(c) Discuss for this particular code if the AR for each of the functions A through D can be allocated statically or not. Explain why or why not.

Problem 2: Activation Records and Stack Layout [30 points]

Under the assumption that the AR are allocated on the stack with the individual layout as shown below, and given the PASCAL code on the right-hand-side answers the following questions:

(a) [20 points] Draw the set of ARs on the stack when the program reaches line 13 in procedure P4. Include all relevant entries in the ARs and use line numbers for the return addresses. Draw direct arcs for the access links and clearly label the values of local variables and parameters in each AR. Also do not forget to include the AR pointer.

(b) [10 points] Explain clearly the values of the access link fields for the instantiation of the procedure P4.




```
01: program main(input,  output);
02:    procedure P1(procedure g(b: integer));
03:       var a: integer;
04:       begin (* P1 *)
05:          a := 3;
06:          g(1);
07:       end (* P1 *)
08:    procedure P2;
09:       var a: integer;
10:       procedure P4(b: integer);
11:          begin (* P4 *)
12:            if(b = 1) then
13:               writeln(b);
14:            else
15:               P4(b-1);
16:          end (* P4 *)
17:       procedure P3;
18:         var a: integer;
19:         begin (* P3 *)
20:           a := 7;
21:           P1(P4)
22:         end (* P3 *)
23:       begin (* P2 *)
24:         a := 0;
25:         P3;
26:       end (* P2 *)
27:  begin (* main *)
28:    P2;
29  end.
```

Problem 3: Procedure Storage Organization [20 points]

Consider the following C program and assume that the ARs follow the same general layout as discussed in problem 2 above. Assuming that an integer requires 4 bytes and a double data type 8 bytes, derive:

a) [10 points] A data layout for the local variables for the AR of the enclosing procedure P detailing the sharing of storage space between variables.
b) [10 points] Outline of the code to access the variable sized arrays assuming indexing of the arrays begins at the index with value 0.

```
B0: {
        int a, b;
        double c;
        ... assign value to a and b
B1:     {
         int v(a), w(b);
        double x;
        … access w(1)
        }
B2:     {
        int f, g, k(b);
         …. access k(2)
        }
    }
```

Problem 4: Register Allocation [40 points]

Consider the following 3-address representation of a computation using scalars and arrays A and B. Assume that no additional registers are needed to access (either writing or reading) the value of an array. As such the access to A[t1] requires no additional registers in addition to the one carrying the value of t1.

```
01:    t1 = i
02:    t2 = A[t1]
03:    t3 = i + 1
04:    t4 = A[t3]
05:    t5 = t2 * t4
06:    t6 = t5 / 2
07:    t7 = i
08:    B[t7] = t6
09:    i = i + 1
```

Questions:

(a) [10 points] Using the bottom-up register allocator described in class assign the various temporary variables and variables to actual registers. For the purpose of this section, assume you have 3 physical registers and that the scalar `i` is already loaded into register r0. At each point when choosing to reuse a register indicate why do you pick each one. Also you should try to reuse registers in copy operations such as t1 = i in line 01 thus not consuming any more registers but simply propagating the use of r0.

(b) [30 points] Use the graph-coloring based algorithm for doing register allocation instead. In this section we are to explore the use of interference webs for different definitions of interference as mentioned in class.

- In the first web use the definition that two variables interfere if there is at least one instruction in which they participate. Derive the interference web between the variables using this definition.
- In the second definition there is no interference if the two webs either do not intersect at all or if they do intersect at a single instruction the web that ends at that instruction participates as the argument of the instruction and the web that begins at that instruction corresponds to the destination value of the instruction.

(c) Using the graph-coloring heuristic described in class with N=3 and determine the coloring and hence register allocation for both interference definitions. Why do they differ, or why not?