CS131 Compilers: Writing Assignment 4
Due Tue, May 31, 2022 at 23:59pm

Tian Haoyuan - 2020533013

This assignment asks you to prepare written answers to questions on semantic analysis. Each of the questions has a short answer. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work. and you should indicate in your submission who you worked with, if applicable. Written assignments are turned in at the start of lecture. You should use the Latex template provided at the course web site to write your solution.

I worked with: nobody

1. **Activation Records.** For each of the variable a, b, c, d, e in this C program, say whether the variable should be kept in memory or a register, and why.

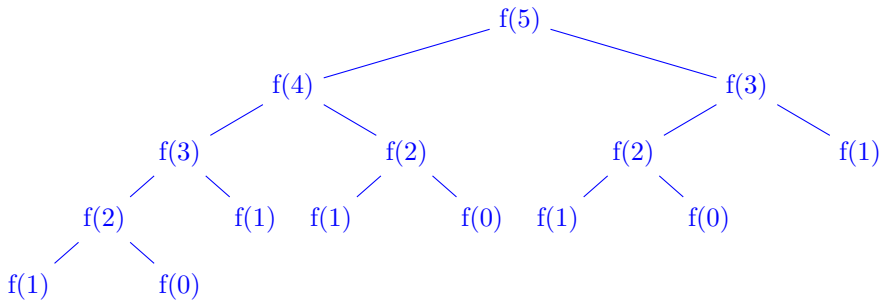
```
int f(int a, int b)
{ int c[3], d, e;
  d = a+1;
  e=g(c,&b);
  return e+c[1]+b;
}
```

a,b,c,d,e should all be kept in memory.
Since a,b are formal parameters that is local to the procedure f(). These are pass by value, so a,b get the copy of real parameters (kept in register), and they are kept in stack.
c,d,e are local variables of the procedure f(). They are kept in stack.

2. **Activation Records.** In the C code to compute Fibonacci numbers recursively. Suppose that the activation record for f includes the following elements in order: (return value, argument n, local s, local t); there will normally be other elements in the activation record as well. The questions below assume that the initial call is f (5).

```
int f(int n) {
    int t, s;
    if (n < 2) return 1;
    s = f(n-1);
    t = f(n-2);
    return s+t;
}
```

(a) Show the complete activation tree.



(b) What does the stack and its activation records look like the first time f(1) is about to return? Let the stack bottom be the top of the table.

$f(5)$retval
$n = 5$
control link
access link
$s = f(4)$
$t = f(3)$
$f(4)$retval
$n = 4$
control link
access link
$s = f(3)$
$t = f(2)$
$f(3)$retval
$n = 3$
control link
access link
$s = f(2)$
$t = f(1)$
$f(2)$retval
$n = 2$
control link
access link
$s = f(1)$
$t = f(0)$
$f(1)$retval=1
$n = 1$
control link
access link

(c) What does the stack and its activation records look like the fifth time f(1) is about to return? Let the stack bottom be the top of the table.

$f(5)$retval
$n = 5$
control link
access link
$s = 5$
$t = f(3)$
$f(3)$retval
$n = 3$
control link
access link
$s = 2$
$t = f(1)$
$f(1)$retval=1
$n = 1$
control link
access link

3. **Code Generation.** Suppose a basic block is formed from the C assignment statements

```
x = a + b + c + d + e + f;
y = a + c + e;
```

(a) Give the three-address statements (only one addition per statement) for this block.
$t_1 = a + b$
$t_2 = t_1 + c$
$t_3 = t_2 + d$
$t_4 = t_3 + e$
$t_5 = t_4 + f$
$x = t_5$
$t_6 = a + c$
$t_7 = t_6 + e$
$y = t_7$

(b) Use the associative and commutative laws to modify the block to use the fewest possible number of instructions, assuming both x and y are live on exit from the block.
$t_1 = a + c$
$t_2 = t_1 + e$
$y = t_7$
$t_3 = t_7 + b$
$t_4 = t_3 + d$
$t_5 = t_4 + f$
$x = t_5$

4. **Run-time environment.** Here is a sketch of two C functions f and g:

```
int f(int x){int i;...return i+1;...}
int g(int y) {int j;...f(j+1). ..}
```

That is, function g calls f . Draw the top of the stack, starting with the activation record for g, after g calls f , and f is about to return. You can consider only return values, parameters, control links, and space for local variables; you do not have to consider stored state or temporary or local values not shown in the code sketch. Answer the following questions:
Let the stack bottom be the top of the table.

(*)
g() retval
int y
(**) control link to (*)
int j
f() retval
int x
(***) control link to (**)
int i

(a) Which function creates the space on the stack for each element?
The caller of g() creates the space of g() retval, int y, control link to (*).
g() creates the space of int j, f() retval, int x, control link to (**).
f() creates the space of int i.

(b) Which function writes the value of each element?
The caller of g() writes the value of control link to(*).
g() writes the value of g() retval, int y, control link to(**), int j.
f() writes the value of f() retval, int x, int i.

(c) To which activation record does the element belong?
int y and int j belong to the activation record of g().
int x and int i belong to the activation record of f().

5. **Code Optimization** Consider the following basic block, in which all variables are integers, and ** denotes exponentiation.

```
a := b + c
z := a ** 2
x := 0 * b
y := b + c
w := y * y
u := x + 3
v := u + w
```

Assume that the only variables that are live at the exit of this block are v and z. In order, apply the following optimizations to this basic block. Show the result of each transformation.

(a) algebraic simplification

```
a := b + c
z := a * a
x := 0
y := b + c
w := y * y
u := x + 3
v := u + w
```

(b) common sub-expression elimination

```
a := b + c
z := a * a
x := 0
y := a
w := y * y
u := x + 3
v := u + w
```

(c) copy propagation

```
a := b + c
z := a * a
x := 0
y := a
w := a * a
u := 0 + 3
v := u + w
```

(d) constant folding

```
a := b + c
z := a * a
x := 0
y := a
w := a * a
u := 3
v := u + w
```

(e) dead code elimination

```
a := b + c
z := a * a
w := a * a
u := 3
v := u + w
```

6. **Code Optimization** Consider the following C code

```
1 int d[10][10];
2 for(int k = 0; k < 10; k++)
3 for(int i = 0; i < 10; i++)
4 for(int j = 0; j < 10; j++)
5 if(d[i][j] > d[i][k] + d[k][j])
6 d[i][j] = d[i][k] + d[k][j];
```

(a) Gives the three-address code for the innermost j loop of the layer.

```
  j = 0
  t1 = j
L1:
  if t1 >= 10 goto L2
  t2 = i * 10
  t3 = t2 + j
  t4 = t3 * 4
  t5 = d[t4]      // t5 is d[i][j]
  t6 = i * 10
  t7 = t6 + k
  t8 = t7 * 4
  t9 = d[t8]      // t9 is d[i][k]
  t10 = k * 10
  t11 = t10 + j
  t12 = t11 * 4
  t13 = d[t12]    // t13 is d[k][j]
  t14 = t9 + t13
  if t5 <= t14 goto L3
  t15 = i * 10
  t16 = t15 + k
  t17 = t16 * 4
  t18 = d[t17]      // t18 is d[i][k]
  t19 = k * 10
  t20 = t19 + j
  t21 = t20 * 4
  t22 = d[t21]      // t22 is d[k][j]
  t23 = t18 + t22
  t24 = i * 10
```

```
  t25 = t24 + j
  t26 = t25 * 4
  d[t26] = t22
L3:
  t27 = j + 1
  j = t27
  goto L1
L2:
```

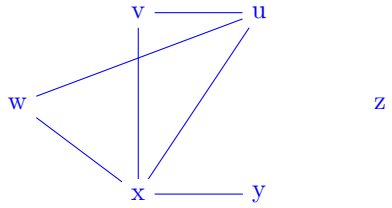
(b) The loop optimization is performed directly on the C source program (including loop-invariant extraction, Inductive variables strength weakening, etc.).

```
int d[10][10];
for(int k = 0; k < 10; k++)
{
    int *temp_k = d[k];
    for(int i = 0; i < 10; i++)
    {
        int temp_ik = d[i][k];
        for(int j = 0; j < 10; j++)
        {
            int temp_sum = temp_ik + temp_k[j];
            if(d[i][j] > temp_sum)
                d[i][j] = temp_sum;
        }
    }
}
```

7. **Register Allocation.** Consider the following program, annotated with live variable information:

```
// live: {v, x}
u = v + 1
// live: {u, v, x}
w = u - v
// live: {u, w, x}
x = x + w
// live: {u, w, x}
y = u - w
// live: {x, y}
z = x + y
// live {z}
```

(a) Draw the interference graph for the program.



(b) What is the smallest number of colors that can be used to color the graph without spilling? Explain why no smaller number of colors will be enough.
The smallest number of colors is 3.
As the largest complete graph in this graph is K3, i.e. the subgraph uvx and edges between them,

also the subgraph wux and edges between them. So clearly uvx are connected to each other, they must all use different colors, so the number of colors can't be less then 3.

(c) Suppose you have machine registers $r1$, $r2$, and $r3$. Write an allocation of variables to registers such that no two adjacent variables have the same register, spilling if necessary.
Let the stack bottom be the bottom of the table.
Construct the stack by each time push a node that is the least connected into the stack, and deleting it's related edges, until all nodes are in the stack.

u
x
w
v
y
z

Then from the top of the stack, distribute a register that would not cause error to the node, and pop the stack, until the stack is empty.

u	r1
x	r2
w	r3
v	r3
y	r1
z	r1

So, an allocation can be $u : r1, x : r2, w : r3, v : r3, y : r1, z : r1$.

8. (10 pts) Consider the following RISCV assembly code program. Using the stack-machine based code generation rules from lecture, what source program produces this code?

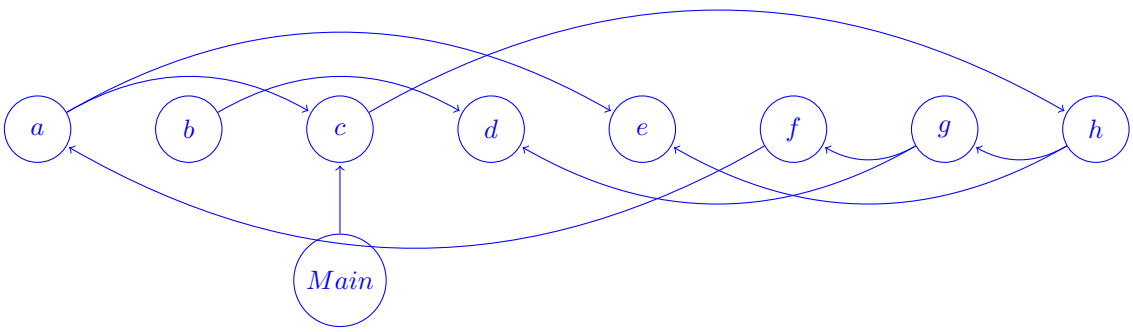
```
f:
  addi    sp,sp,-48
  sw      s0,44(sp)
  addi    s0,sp,48
  sw      a0,-36(s0)
  sw      a1,-40(s0)
  sw      zero,-20(s0)
  sw      zero,-24(s0)
  j       .L2
.L3:
  lw      a5,-24(s0)
  slli    a5,a5,2
  lw      a4,-36(s0)
  add     a5,a4,a5
  lw      a5,0(a5)
  lw      a4,-20(s0)
  add     a5,a4,a5
  sw      a5,-20(s0)
  lw      a5,-24(s0)
  addi    a5,a5,1
  sw      a5,-24(s0)
.L2:
  lw      a4,-24(s0)
  lw      a5,-40(s0)
  blt     a4,a5,.L3
  lw      a5,-20(s0)
```

```
  mv      a0,a5
  lw      s0,44(sp)
  addi    sp,sp,48
  jr      ra

int f(int *arr, int n)
{
    int sum;
    int i;
    for (i = 0; i < n; ++i)
    {
        sum += arr[i];
    }
    return sum;
}
```

9. (10*3=30 pts) Consider the following Cool program:

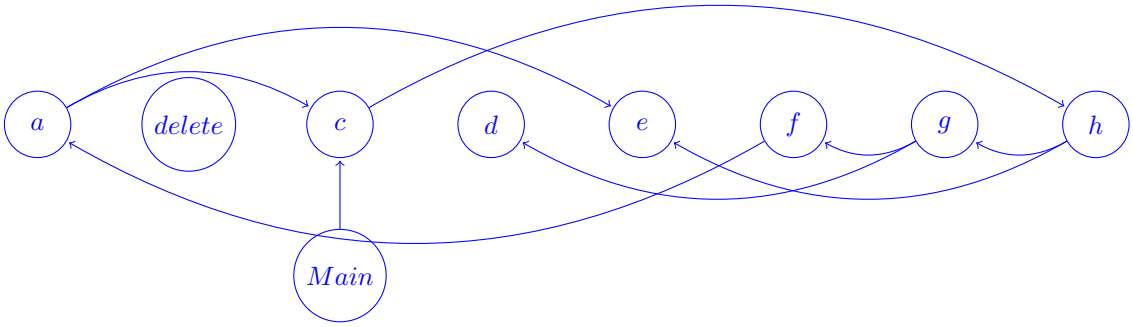
```
class C {
  x : C; y : C;
  setx(newx : C) : C { x <- newx };
  sety(newy : C) : C { y <- newy };
  setxy(newx : C, newy :C) : SELF_TYPE {{ x <- newx; y <- newy; self; }};
};

class Main {
  x:C;
  main() : Object {
    let a : C <- new C, b :C <- new C, c : C<- new C, d : C <- new C,
    e : C <- new C, f :C <- new C, g : C <- new C, h : C <- new C in {
      f.sety(a), a.setxy(e, c); b.setx(d); g.setxy(d,f); c.sety(h); h.setxy(e, g); x <- c;
    }
  };
};
```

(a) (10 pts) Draw the heap at the end of execution of the above program, identifying objects by the variable names to which they are bound in the let expression. Assume that the root is the Main object created at the start of the program, and this object is not in the heap (note that Main is pointing to c).

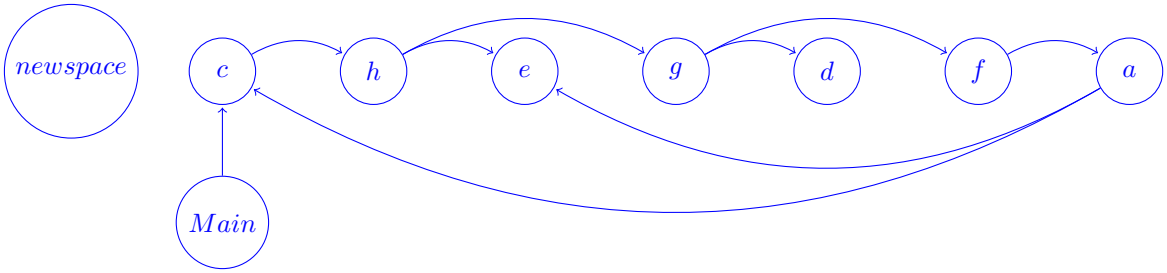


(b) (10 pts) For each of the garbage collection algorithms discussed in class (Mark and Sweep, Copy Collector, Reference Counting), show the heap after garbage collection.

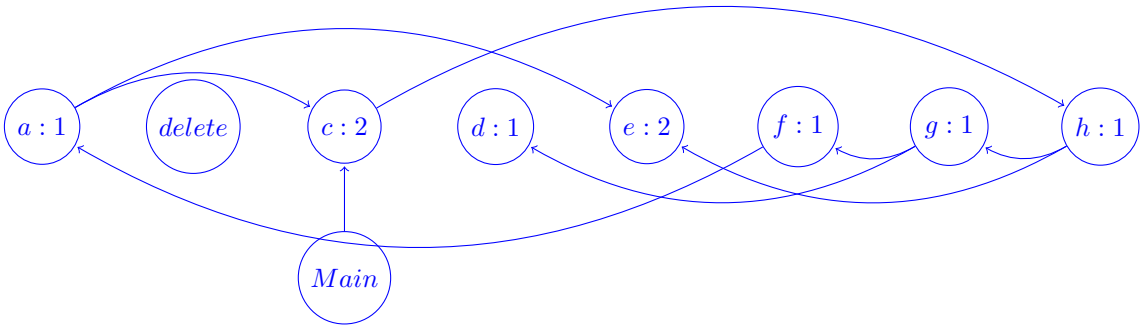
Mark and Sweep



Copy Collector



Reference Counting



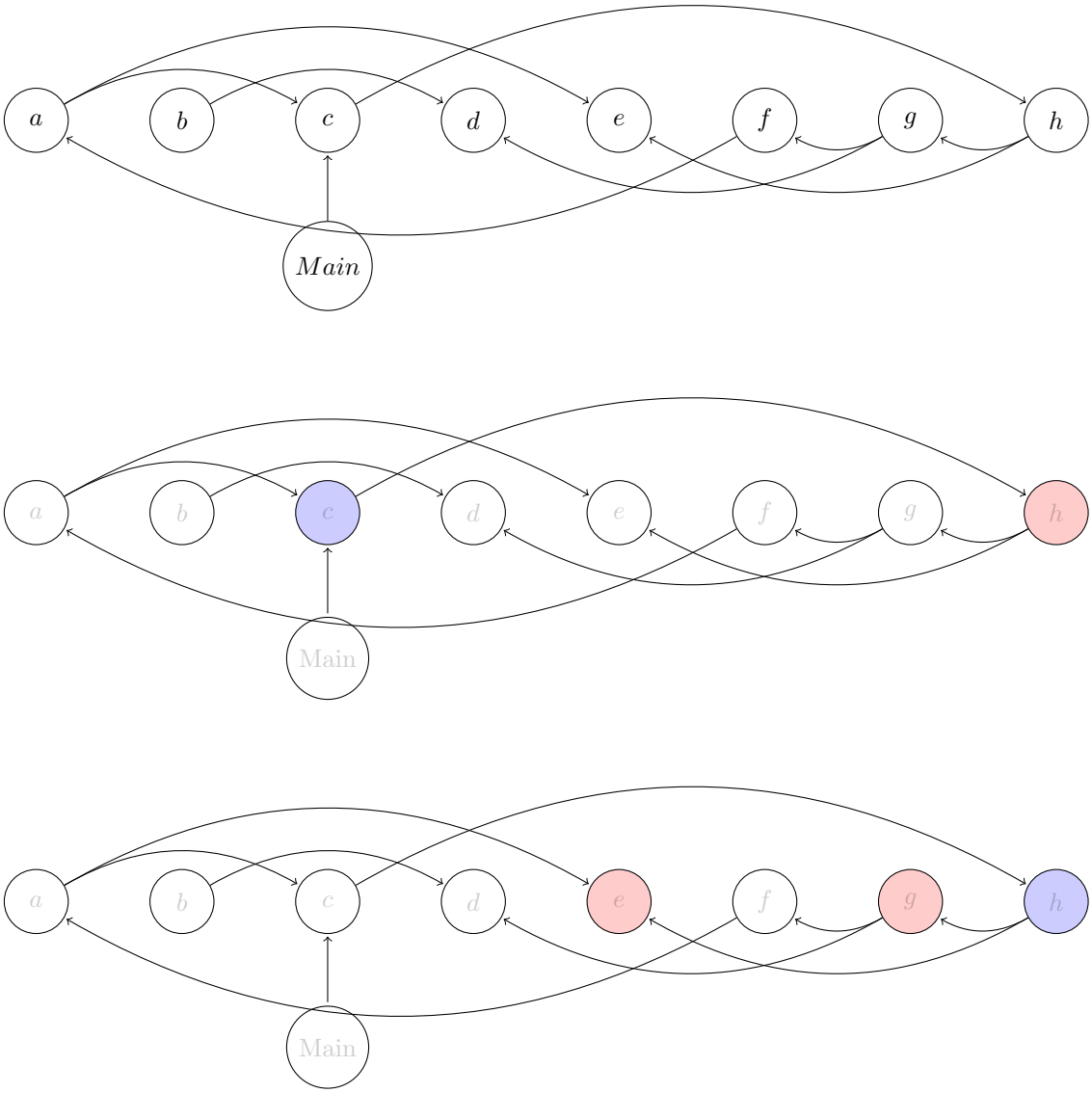
(c) (10 pts) Suppose the cost c_i is every reachable words' instruction number, comapare 3 algorithms, let H be the size of the heap, let R be the reachable word's data size, when will the Mark and Sweep outperform Copy Collector.

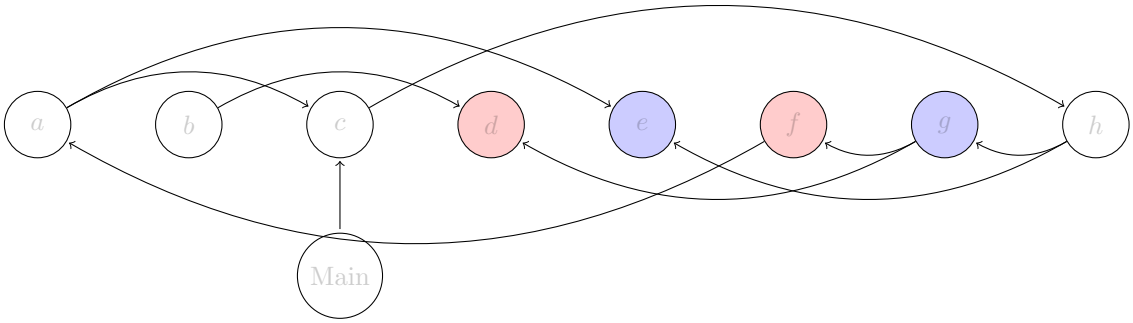
As for memory performance the Mark and Sweep algorithm is O(H), while the Copy Collector algorithm is O(R+H). So the former one outerperforms the latter one when considering the memory cost.

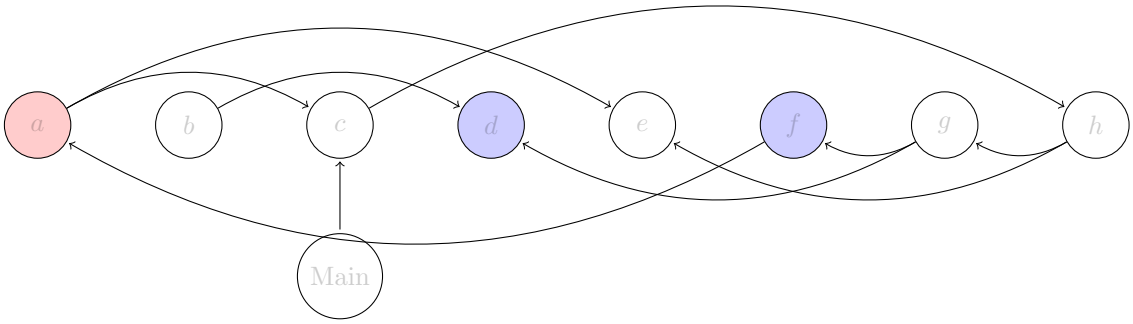
As for time performance, the former algorithm is $O(R^2 + H)$, while the latter one is $O(R^2)$. But when H is not too bigger than R, in which case not too little words are reachable, so that we can omit $O(H)$, and get prettry the same time performance. Still, we have to consider the constant coefficient of both time performance. Mark and Sweep is about $O(2R^2)$, and considering

the pseudo code, $c_i = 4$. While for Copy Collector, the copy processes comsume plenty of time, comparing to the formal one, despite the have similar c_i, the mark and sweep should outerperforms the latter one by constant multiples.

(d) (10 pts) Suppose the marking process is multithreaded, use the above example show that tricolor marking with write barrier is thread safe.




a
b
c
d
e
f
g
h
Main


a
b
c
d
e
f
g
h
Main