CS 132 Compiler Construction, Fall 2012

Instructor: Jens Palsberg

Multiple Choice Exam, Dec 13, 2012

ID

Name

This exam consists of 22 questions. Each question has four options, exactly one of which is correct, while the other three options are incorrect. For each question, you can check multiple options.

I will grade each question in the following way. If you check *none* of the options, you get 0 points. If you check all *four* options, you get 0 points.

Check one option. If you check *one* option, and that option is correct, you get 2 points. If you check *one* option, and that option is wrong, you get –0.667 points (yes, negative!).

Check two options. If you check *two* options, and one of those options is correct, you get 1 point. If you check *two* options, and both of them are wrong, you get –1 point (yes, negative!).

Check three options. If you check *three* options, and one of those options is correct, you get 0.415 points. If you check *three* options, and all three of them are wrong, you get –1.245 points (yes, negative!).

The maximum point total is $22 \times 2 = 44$ points. I will calculate a percentage based on the points in the following way:

$$\frac{\max(0, \text{point total})}{44} \times 100$$

Notice that if your point total is negative, you will get 0 percent.

Example
Consider the following Java program:

```
class C {
  C x;
  int f;
  void run(boolean b, int x) {
    C y;
    y = new C();
    if (b) { this.bump(y,x); }
    else   { f = this.x.f + 1; }
  }

  void bump(C z, int j) {
    z.f=j;
  }
}
```

Question 1
Does the program type check?

- *a* ☐ Yes
- *b* ☐ No
- *c* ☐ The question cannot be answered based on the given information.
- *d* ☐ A Java type checker would throw an exception.

Question 2

What is the symboltable at the point where the type checker will try to type check the statement `f = this.x.f +1;`.

a ☐ The following symboltable in which we search for identifiers from the bottom:

Id	Type
b	boolean
x	int
y	C

b ☐ The following symboltable in which we search for identifiers from the top:

Id	Type
b	boolean
x	int
y	C

c ☐ The following symboltable in which we search for identifiers from the bottom:

Id	Type
x	C
f	int
b	boolean
x	int
y	C

d ☐ The following symboltable in which we search for identifiers from the top:

Id	Type
x	C
f	int
b	boolean
x	int
y	C

Question 3

Consider the translation of the method `run` to intermediate code in the style of Vapor. What is reasonable Vapor-style code for `y = new C();` ?

a ☐
```
y = heapallocate 12
t.1 = heapallocate 8
store y + 0 := t.1
store t.1 + 0 := C.run
store t.1 + 4 := C.bump
```

b ☐
```
y = heapallocate 8
t.1 = heapallocate 12
store y + 4 := t.1
store t.1 + 0 := C.run
store t.1 + 4 := C.bump
```

c ☐
```
y = heapallocate 12
t.1 = heapallocate 8
store y + 0 := t.1
store t.1 + 4 := C.run
store t.1 + 8 := C.bump
```

d ☐
```
y = heapallocate 8
t.1 = heapallocate 12
store y + 4 := t.1
store t.1 + 4 := C.run
store t.1 + 8 := C.bump
```

Question 4

Consider the translation of the method `run` to intermediate code in the style of Vapor. What is reasonable Vapor-style code for `this.bump(y,x);` ?

a ☐
```
load t.2 := this + 0
load t.3 := t.2 + 4
call t.3(this, y, x)
```

b ☐
```
load t.2 := this + 0
load t.3 := t.2 + 4
call t.3(t.2, y, x)
```

c ☐
```
load t.2 := this + 0
load t.3 := t.2 + 0
call t.3(this, y, x)
```

d ☐
```
load t.2 := this + 0
load t.3 := t.2 + 0
call t.3(t.2, y, x)
```

Question 5

Consider the translation of the method `run` to intermediate code in the style of Vapor. What is reasonable Vapor-style code for `f = this.x.f + 1;` ?

a ☐
```
load t.4 := this + 0
load t.5 := t.4 + 4
move t.6 := t.5 + 1
store this + 4 := t.6
```

b ☐
```
load t.4 := this + 4
load t.5 := t.4 + 0
move t.6 := t.5 + 1
store this + 4 := t.6
```

c ☐
```
load t.4 := this + 0
load t.5 := t.4 + 4
move t.6 := t.4 + 1
store this + 4 := t.5
```

d ☐
```
load t.4 := this + 4
load t.5 := t.4 + 0
move t.6 := t.4 + 1
store this + 4 := t.5
```

Question 6

Consider the translation of the method `run` to intermediate code in the style of Vapor. Let S_1 denote `this.bump(y,x);`, and let S_2 denote `f = this.x.f + 1;`. Assume that we represent true by 1 and that we represent false by 0. What is reasonable Vapor-style code for `if (b)` S_1 `else` S_2 ?

a ☐

```
        if0 b goto :else
        translation of S1
        goto :end
else:   translation of S2
end:
```

b ☐

```
        if b goto :else
        translation of S1
        goto :end
else:   translation of S2
end:
```

c ☐

```
        if0 b goto :else
        translation of S2
        goto :end
else:   translation of S1
end:
```

d ☐

```
        if b goto :else
        translation of S2
        goto :end
else:   translation of S1
end:
```

Question 7

Assume that the compiler does not map any temporaries, variables, or arguments to registers. What is the MIPS stack layout for the method `run` in class `C` ? In each of the possible answers, the stack pointer points to bottom element.

a ☐
`b`
`x`
return address
`y`
temporaries generated by the compiler
saved registers

b ☐
`b`
`x`
return address
`y`
temporaries generated by the compiler
saved registers
`x`
`f`

c ☐
`b`
`x`
return address
`y`
temporaries generated by the compiler

d ☐
`b`
`x`
return address
`y`
temporaries generated by the compiler
`x`
`f`

Example
Consider the following program that uses the variables `b`, `i`, and `y`:

```
L1:  i = 1
L2:  y = i + 2
L3:  if y > 10 then goto L9
L4:  b = y + 3
L5:  if i > 20 goto L7
L6:  i = i + 1
L7:  b = b + i
L8:  goto L2
L9:  y = b + 4
L10: return y
```

Question 8
In the control-flow graph for the program, which variables are live along any of the edges to L1 ?
- a ☐ `b`
- b ☐ `y`
- c ☐ `i`
- d ☐ None

Question 9
In the control-flow graph for the program, which variables are live along any of the edges to L2 ?
- a ☐ `b, i`
- b ☐ `b, y`
- c ☐ `i, y`
- d ☐ `b, i, y`

Question 10
In the control-flow graph for the program, which variables are live along any of the edges to L3 ?
- a ☐ `i, y`
- b ☐ `b, i`
- c ☐ `b, y`
- d ☐ `b, i, y`

Question 11
In the control-flow graph for the program, which variables are live along any of the edges to L4 ?
- a ☐ `b, i, y`
- b ☐ `b, i`
- c ☐ `b, y`
- d ☐ `i, y`

Question 12

In the control-flow graph for the program, which variables are live along any of the edges to L5 ?

- a ☐ `b, i, y`
- b ☐ `b, i`
- c ☐ `b, y`
- d ☐ `i, y`

Question 13

In the control-flow graph for the program, which variables are live along any of the edges to L6 ?

- a ☐ `b, i`
- b ☐ `b, y`
- c ☐ `i, y`
- d ☐ `b, i, y`

Question 14

In the control-flow graph for the program, which variables are live along any of the edges to L7 ?

- a ☐ `b, i, y`
- b ☐ `b, i`
- c ☐ `b, y`
- d ☐ `i, y`

Question 15

In the control-flow graph for the program, which variables are live along any of the edges to L8 ?

- a ☐ `b, y`
- b ☐ `b, i`
- c ☐ `i, y`
- d ☐ `b, i, y`

Question 16

In the control-flow graph for the program, which variables are live along any of the edges to L10 ?

- a ☐ `y`
- b ☐ `i`
- c ☐ `b`
- d ☐ None

Example
The following is the body of an entire procedure that uses six temporaries `a`, `b`, `c`, `d`, `e`, `f`.

```
a = 1
b = 2
c = a + b
d = a + c
e = c + d
f = d + 3
b = e + 4
c = b + e
f = b + f
b = c + f
return b + 5
```

Question 17
Assume that we must assign one register to each of the six variables. What is the fewest number of registers that is needed for this program, *without* spilling?

a ☐ 3
b ☐ 4
c ☐ 5
d ☐ 6

Question 18
Assume that four registers `r1,r2,r3,r4` are available, and that we run the linear scan register allocation algorithm on the program. Which variables get spilled to memory?

a ☐ None
b ☐ `b`
c ☐ `c`
d ☐ `f`

Question 19
Assume that three registers `r1,r2,r3` are available, and that we run the linear scan register allocation algorithm on the program. Which variables get spilled to memory?

a ☐ `b`
b ☐ None
c ☐ `c`
d ☐ `f`

Question 20
Assume that two registers `r1,r2` are available, and that we run the linear scan register allocation algorithm on the program. Which variables get spilled to memory?

a ☐ `b,c`
b ☐ `b`
c ☐ `c`
d ☐ `b,f`

Example
The following fragment of the above procedure uses four temporaries `a`, `b`, `c`, `d`:

```
b = 2
c = a + b
d = a + c
```

Assume that we have done register allocation for the entire procedure that contains this program fragment, and that we have assigned `a` to register `r1`, `b` to register `r2`, `c` to a memory location that we denote by `c`, and `d` to a memory location that we denote by `d`. Assume also that we have a spare register `v0`.

Question 21
What is correct code for the program fragment?

a ☐
```
move r2 := 2
move v0 := r1 + r2
store c := v0
move v0 := r1 + v0
store d := v0
```

b ☐
```
move r2 := 2
move v0 := r1 + r2
store c := v0
move v0 := r1 + r2
store d := v0
```

c ☐
```
move r2 := 2
move v0 := r1 + r2
store c := r1
move v0 := r1 + v0
store d := v0
```

d ☐
```
move r2 := 2
move v0 := r1 + r2
store c := v0
move v0 := r1 + v0
store d := r2
```

Example
The following fragment of the above procedure uses three temporaries `b`, `c`, `f`:

```
f = b + f
b = c + f
```

Assume that we have done register allocation for the entire procedure that contains this program fragment, and that we have assigned `b` to register `r1`, `c` to a memory location that we denote by `c`, and `f` to a memory location that we denote by `f`. Assume also that we have spare registers `v0` and `v1`.

Question 22
What is correct code for the program fragment?

a ☐
```
load v0 := f
move v0 := r1 + v0
store f := v0
load r1 := c
move r1 := r1 + v0
```

b ☐
```
load v0 := f
move v0 := r1 + v0
store f := v0
load v1 := c
move r1 := r1 + v0
```

c ☐
```
load v0 := f
move v0 := r1 + v0
store f := v0
load v1 := c
move r1 := r1 + v1
```

d ☐
```
load v0 := f
move v0 := r1 + v0
store f := v0
load r1 := f
move r1 := r1 + v0
```