(ii) let x = 1
print(x)
x := "abc"
print(x)

valid with dynamic types
invalid with static types

p64 Q
AM

Solution Notes Compiler Construction 2004

[Syllabus: "variable binding [and tree based interpreter]" and "implementation of a section of interesting things [exceptions]"]

(a) Static binding looks up variables in the env of the definer, whereas dynamic binding looks them up the in the env of the caller. Consider

```
let x = 1 in
let f() = x in
let g(x) = f() in
print f(2)
```

which prints 1 using static scoping and 2 using dynamic scoping.

(b) Consider

```
class C { Cmembers };
class D extends C { Dmembers };
```

Variables in Java are really pointers, and we exploit the fact that the effective members of D are *Cmembers* $\cup$ *Dmembers* by arranging for the storage layout for D to initially be exactly that for C followed by *Dmembers*. Thus a D can be passed to a C and still look like a C to the callee. The reverse process is generally unsafe, e.g. attempting use a C as if it were a D could lead to referencing memory beyond the allocated size (or even to a wrongly typed member if C were also extended to make `class E`. Therefore at runtime the "upward cast" operation

```
D d = (D)c
```

needs a run-time check to ensure that the value being cast is indeed (i.e. has the correct storage layout for) a D (presumably D (or further extended D) previously stored (as above) in a variable of type C.

(c) This is just a simpler version of exceptions. Normally exceptions are dynamically scoped, so that on entry to `try` block we push a new label (here I mean label closure, i.e. code point together with frame pointer [and in some implementations stack (fringe) pointer too]) on an exception stack H, and pop it on normal exit. If an exception occurs then the top item of H is popped and branched to (there are irrelevant issues here as to whether the exception type is checked before branching to it, or by checking it on entry and doing a re-raise).

However, here all we need to do is:

- On encountering a `try` block, store the label closure (stack pointer+label) on the stack as if it were a statically bound variable.
- On exiting a `try` block, no action needs to be taken

- On an exception load (via the static chain) the stack pointer into FP/SP, and branch to the label.

Clearly even this is overkill, and there's no point in storing a stack pointer in the stack frame which contains it, but at least it simplifies the explanation!