Part B Long questions

B1) [20 points]
The front end of the YES compiler you wrote for this class consists of four phases: the scanner, the parser, the name analyzer, and the type checker. At each phase, different errors are identified. Behind this page is a YES program that contains multiple errors. Specific lines in the code are indicated with a location number using a comment in the form `// loc N` (where `N` is a number).

For each line of code marked with a location, you are to consider whether that line contains an error that would be identified by one of the phases of the front end of the YES compiler. If the line contains no errors, write "*No error*". If the line does contain an error, briefly describe what the error is and indicate which phase (scanner, parser, name analyzer, type checker) is responsible for identifying and reporting that error. Location 0 is done for you as an example. Note: consider each location without regard for errors that may have occurred at earlier locations. For example, report type checker errors even if the program would have already been rejected by the parser.

Loc	**Error description**	**Phase**
0	Unterminated string literal	Scanner
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

```
void printGreeting() {
      cout << "Hello!  // loc 0
}
void fctnA(int a, bool b) {
      if (b) {
            cin >> "value";  // loc 1
      }
      fctnB();          // loc 2
}
struct S {
      int S;                 // loc 3
};
struct T {
      bool b;
};
void fctnB(int x, int y) {
      bool x;                // loc 4
      cout << "y = " << y;  // loc 5
}
bool fctnC(int a, int b, int c) {
      if (a = b) {      // loc 6
            b = 6;
      }
      return (a + b < c);
}
void main() {
      int x;
      int y;
      y = 5;
      if (y >= -4) {
            struct T x;      // loc 7
            int p;
            fctnA(8, false);
      }
      else {
            x.y.b = true;    // loc 8
            p = 5;           // loc 9
      }
      z = fctnB(6);     // loc 10
}
```

B2) [20 points]
Consider the following incomplete code

```
int x=5, y=20, z=50;

void f(int a, int b) {
    a = a + 1;
    x = x + 2;
    y = y + 3;
    <var 1> = <var 1> + <var 2>;
}

void main() {
    f( <actual 1>, <actual 2> );
    print(x, y, z);
}
```

The table below shows the values printed for *x*, *y*, and *z*, assuming the given parameter-passing modes for each of *f*'s parameters, *a* and *b*.

Param-passing modes		**Output**		
a	*b*	*x*	*y*	*z*
value	value	7	23	75
reference	value	8	23	75
name	name	8	23	81
value-result	value	6	23	75

What *single* replacement could be used for each of the placeholders so that the completed code produces the output specified above for each of the four pairs of parameter-passing modes? (For partial credit, give replacements that work for at least two of the four cases; say which cases don't work, and give the outputs that would be produced for those cases.)

< var 1 > : ☐ ***< actual 1 > :*** ☐

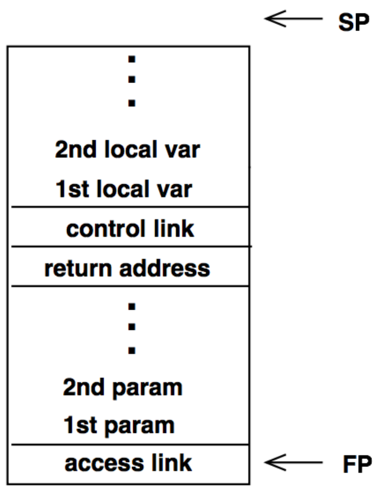
< var 2 > : ☐ ***< actual 2 > :*** ☐

B3) [15 points]

Assume we have a language that allows nested procedures, and both value and reference parameters. Below is an example program written in that language. Note that there are three reference parameters (indicated using ampersands, as is done in C++): *x* in procedure *f* , *a* in procedure *g*, and *w* in procedure *h*.

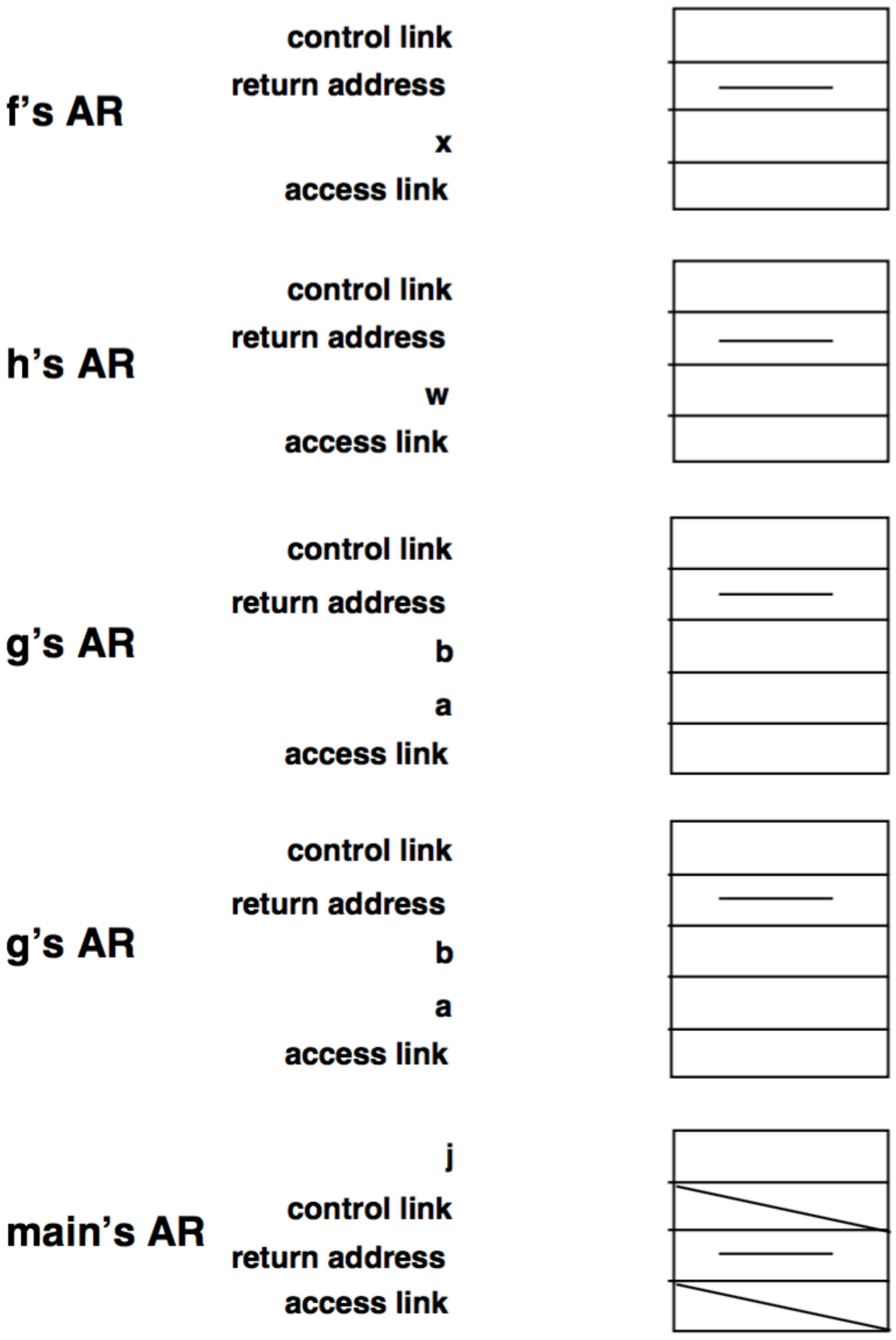
Assume that access to non-local variables is implemented using *access links* and that Activation Records are organized as shown in the picture below next to the code.

```
void main() {
    int j=1;

    void f(int & x) {
        ...
    }
    void g(int & a, int b) {

        void h(int & w) {
            f(a);
        }

        // body of g
        j++;
        if (j==2) g(b, a);
        h(j);
    }
    // body of main
    g(j, j);
}
```



Fill in the activation records on the next page as they would be when the body of procedure *f* is executing. Note that you need not fill in the *return-address* fields, and that the access and control links for *main*'s activation record are already filled in with nulls.

To make your picture easier to read, fill in the control links on the left side of the stack, and fill in the access links and other pointers on the right.


f's AR
control link
return address
x
access link
h's AR
control link
return address
w
access link
g's AR
control link
return address
b
a
access link
g's AR
control link
return address
b
a
access link
main's AR
j
control link
return address
access link

B5) [10 points]

Part a)

One of the optimizations we discussed was moving the computations of loop-invariant expressions out of their loops (LICM). For example, if the assignment statement x=y+z is in a loop, and neither y nor z can be changed inside the loop, then we add a new assignment tmp1=y+z in the loop's pre-header, and we replace x=y+z inside the loop with x=tmp1.

Give a simple example to illustrate why it would not, in general, be correct to move the whole assignment statement outside the loop (because it could change the program's behavior). Explain your example.

Part b)

Assume you are trying to optimize the following sequence of code by removing redundant store instructions:

```
sw $t0, -4($sp)
sw $t0, -4($sp)
lw $t1, -4($sp)
```

Is it necessarily safe to remove one of the store instructions if this sequence of code all belongs to the same basic block?

Is it necessarily safe to remove one of the store instructions if this sequence does NOT all belong to the same basic block?

Justify your answers