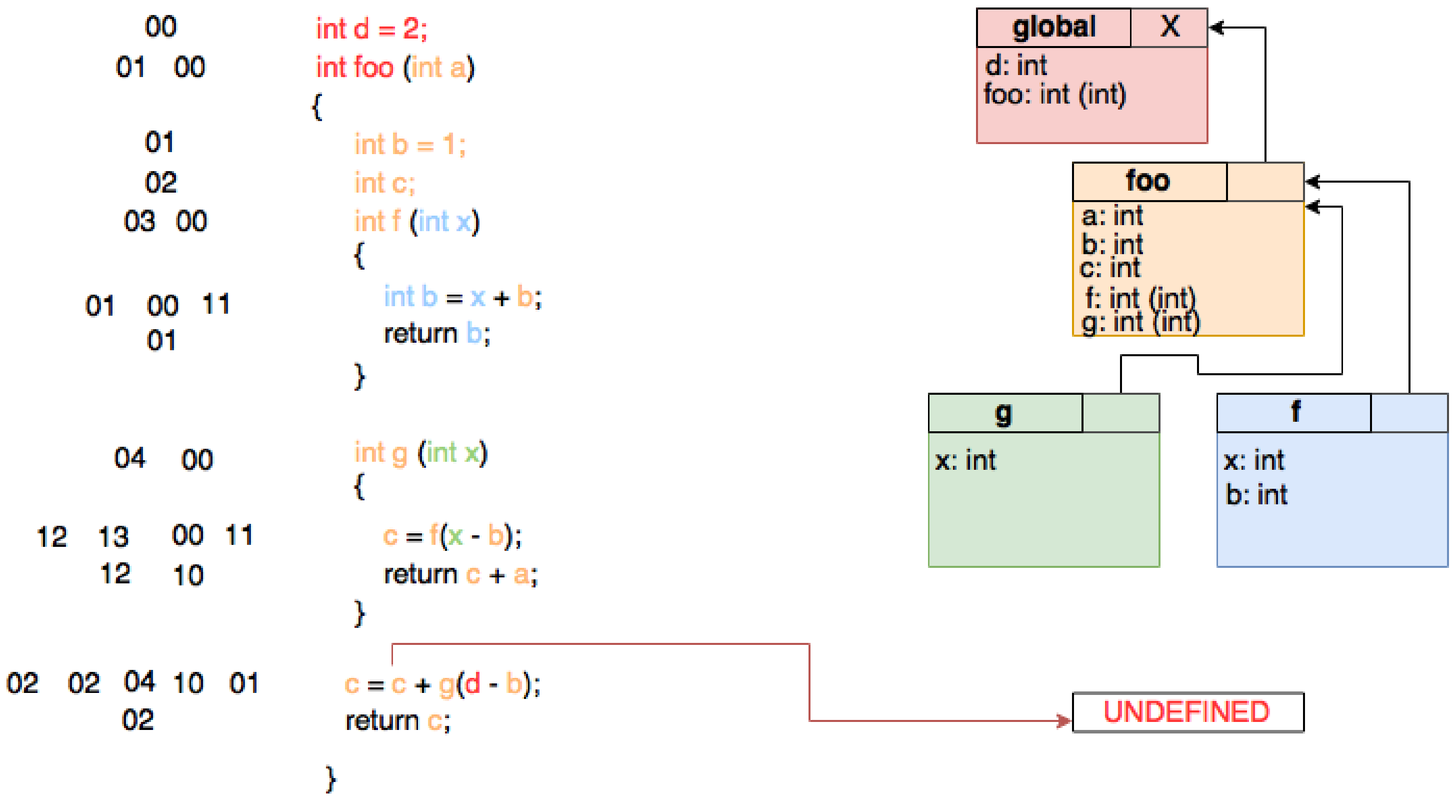
Assignment Series 4

Semantic Analysis
Andrea & Aynel

Assignment 15: Scoping and symbol tables



```
int d = 2;
int foo (int a)
{
    int b = 1;
    int c;
    int f (int x)
    {
        int b = x + b;
        return b;
    }

    int g (int x)
    {
        c = f(x - b);
        return c + a;
    }

    c = c + g(d - b);
    return c;

}
```

a. It is difficult to give an answer, because it is is undefined (see diagram). It is not clear what side of the addition will be valued first.

If the left side is valued first, this would give an error, because c does not have a value yet (**int c;**). However, if the right side is being valued first, (global) c will be updated in the other functions and will have a value of 1 (because **c = f(1 - 1);** and **int b = 0 + 1;**). The answer to foo(8) then will be **c = 1 + 9**).

Moreover, in int f(int x) the second b is valued as 1, because variables are only in scope after declaration.

b. Used colors to map the code to the scope

c + d. All annotated in the diagram.

Assignment 16: Lambda lifting

Original:

```
int d = 2;
int foo ( int a)
{
        int b = 1;
        int c;

        int f( int x) {
                int b = x + b;
                return b;
        }

        int g( int x) {
                c = f( x - b);
                return c + a;
        }

        c = c + g( d - b);
        return c;
}
```

New:

```
int d = 2;
int f_inner (int x, int b) {
        int h = x + b;
        return h;
}

int g_inner ( int x, int b, int *c, int a) {
        *c = f_inner( x - b);
        return *c + a;
}

int foo (int a)
{
        int b = 1;
        int c;
        c = c + g_inner( d – b, b, &c, a);
        return c;
}
```

Assignment 17: Function overloading

int foo (int a) {}
int foo (float a) {}
int foo (int a, int b) {}
int foo (float a, float b) {}

While doing the semantic analysis, it would become difficult to dispatch to the matching function definition. You would have to look at the inferred argument types as well to distinguish between the functions with the same name. When implementing function overloading you could resolve this problem by changing the names of the same functions within a scope, using type analysis. See example below.

```
int foo_int (int a) {}
int foo_float (float a) {}
int foo_int_int (int a, int b) {}
int foo_float_float(float a, float b) {}
```