Concepts of Programming Languages
Computer Science 141
Fall 2015

Notes: The C3 Language and Parameter Passing
Prepared by: Tatiana Bradley
More info: See the lecture slides / Ghezzi handout (especially the sections on C3 and parameter passing)

The C3 Language

C3 in context

Recall the C2 language, which allowed us to do simple function calls, with no recursion or return values. For each function, we had a segment of the code array that held the instructions for that function, and a segment of the data array (called an "Activation Record" or AR) that held the data associated with the function (i.e., the return pointer and the local variables). We could hold the AR at a fixed position in the data array because we knew there could never be two instances of the same function active at the same time.

In contrast, when translating C3 into SIMPLESEM, we don't have the luxury of storing the AR of a function in a fixed location. This is because a function can recursively call itself, and we could have arbitrarily many instances of the same function active at the same time. Thus we need extra bookkeeping values to tell us where the AR is inside the data array, and how to get back to the previous function when we are done with the current one.

Frame Pointer, Stack Pointer, and Activation Records in C3

Since we don't know until runtime where an AR will live inside the data array, we need two special pieces of data to keep track of it:

1. Frame pointer (fp). The frame pointer is the address of the first cell of the current activation record. By Ghezzi's convention, we store the frame pointer in D[0].

2. Stack pointer (sp). The stack pointer is the address of the empty cell directly following the current activation record. By Ghezzi's convention, we store the stack pointer in D[1].

In C3, activation records hold the following information:

1. Return Pointer (RP). The instruction to jump to when the function is finished.
2. Dynamic Link (DL). The frame pointer to restore when the function is finished, i.e., the frame pointer of the caller.
3. Arguments (optional) - the parameters passed into the function. More on this in the Parameter Passing section of this document.
4. Local variables
5. Return value (RV) - will only be here if this function has called another function with a return value.

Note on return values: When a function calls another function that returns a value, the calling function extends its AR by one cell to store the return value.

Calling and returning from functions in C3

How can we implement this functionality in SIMPLESEM? We need to make sure that we set everything up for the callee when we call a function, and then clean up properly when we return. There is SIMPLESEM code for this on pg 102 of the Ghezzi handout.

Notation:
In the following, let `sp` and `fp` denote the memory addresses of the stack pointer and frame pointer, respectively. By Ghezzi's convention, these will be `fp = 0` and `sp = 1`. Then `D[fp]` means the current value of the frame pointer, and `D[D[fp]]` means the value that the frame pointer is pointing to (i.e, the contents of the first cell of the active function's AR).

Set up (by the caller):

1. (If the function we are calling has a return value). Allocate a data cell in current AR for the return value. Since the stack pointer tells us where our AR ends, set `D[sp] = D[sp] + 1` to add one cell to our AR.

2. Set the RP in the callee's activation record. The current stack pointer is the same as the callee's frame pointer, so set `D[D[sp]]] = RP`. RP is the address (in the code array) of the first instruction we want to execute after we return from the called function.

3. Set the DL in the callee's activation record. Set `D[ D[sp] + 1 ] = D[fp]`.

4. Reset the frame pointer, i.e., set `D[fp] = D[sp]`.

5. Reset the stack pointer. Set `D[sp] = D[sp] + size of AR`. The size of the AR is 2 + # args + # local vars.

6. Jump to the first instruction in the callee's code.

Clean up (by the callee):

1. If the callee returns a value: set `D[ D[fp] - 1 ]` equal to the desired return value. (This is modifying the last value of the caller's AR).

2. Restore the stack pointer. Set `D[sp] = D[fp]`.

3. Restore the frame pointer. Set `D[fp] = DL`. The dynamic link is found at the memory cell `D[fp] + 1` (before `D[fp]` is reset).

4. Jump back to the caller by grabbing the RP from `D[ D[sp] ]`.

Parameter Passing

Pass-by-Value vs Pass-by-Reference

When a parameter is passed by value, the calling function gives a copy of the parameter's value to the called function. When a parameter is passed by reference, the calling function gives a copy of the *address* of the parameter to the called function.
As an example, consider the following pseudocode:

```
main:
     int x = 5
     function(x)
     print x
function(y):
     y = 2
```

If `x` is passed to `function` by value, then the program prints `5`. In contrast, if `x` is passed to `function` by reference, then the program prints `2`.

Parameter passing in SIMPLESEM

To implement parameter passing, we need more space in our AR for arguments. Arguments live under the RP, and DL, but above the local variables. The calling function should write the parameters (or the parameter addresses in the case of pass-by-reference) to the appropriate cells in the callee's AR in the setup phase of calling a function.

Note: If we are passing a parameter that is not a variable by reference (for example, an expression or a constant like `5` or `x+1`, that does not have a place in memory), we need to store that value in the caller's AR as a temporary variable in the local variable section. We then send the address of the temporary cell to the callee.