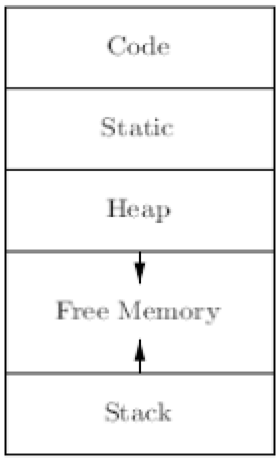
STORAGE ORGANIZATION

The management and organization of this logical address space is shared between the compiler, operating system, and target machine. The operating system maps the logical addresses into physical addresses, which are usually spread throughout memory.

The run-time representation of an object program in the logical address space consists of data and program areas as shown in figure. A compiler for a language like C++ on an operating system like Linux might subdivide memory as in figure.



- CODE AREA: The size of the generated target code is fixed at compile time, so the compiler places the executable target code in a statically determined area, the low end of memory.
- STATIC AREA: Size of some program data objects, such as global constants, and data generated by the compiler, such as information to support garbage collection, may be known at compile time, and these data objects can be placed in another statically determined area. In early versions of Fortran, all data objects could be allocated statically.
- STACK AND HEAP AREA: To maximize the utilization of space at run time, the other two areas, Stack and Heap, are at the opposite ends of the remainder of the address space. These areas are dynamic; their size can change as the program executes. These areas grow towards each other as needed.
- STACK AREA: Used to store data structures called activation records that get generated during procedure calls.
- An activation record is used to store information about the status of the machine, such as the value of the program counter and machine registers, when a procedure call occurs. When control returns from the call, the activation of the calling procedure can be restarted after restoring the values of relevant registers and setting the program counter to the point immediately after the call. Data objects whose lifetimes are contained in that of activation can be allocated on the stack along with other information associated with the activation.
- HEAP AREA: Many programming languages allow the programmer to allocate and deallocate data under program control. For example, C has the functions malloc and free that can be used to obtain and give back arbitrary chunks of storage. The heap is used to manage this kind of long-lived data.

STACK ALLOCATION SPACE

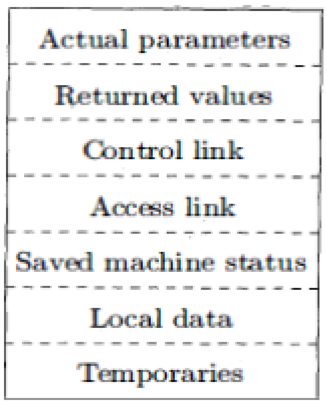
- Each time a procedure is called, space for its local variables is pushed onto a stack, and when the procedure terminates, that space is popped onto the stack.
- This arrangement not only allows space to be shared by procedure calls whose durations do not overlap in time, but it allows us to compile code for a procedure in such a way that the relative addresses of its nonlocal variables are always the same, regardless of the sequence of procedure calls.

Activation Trees

Stack allocation would not be feasible if procedure calls, or activations of procedures, did not nest in time.

Activation Records

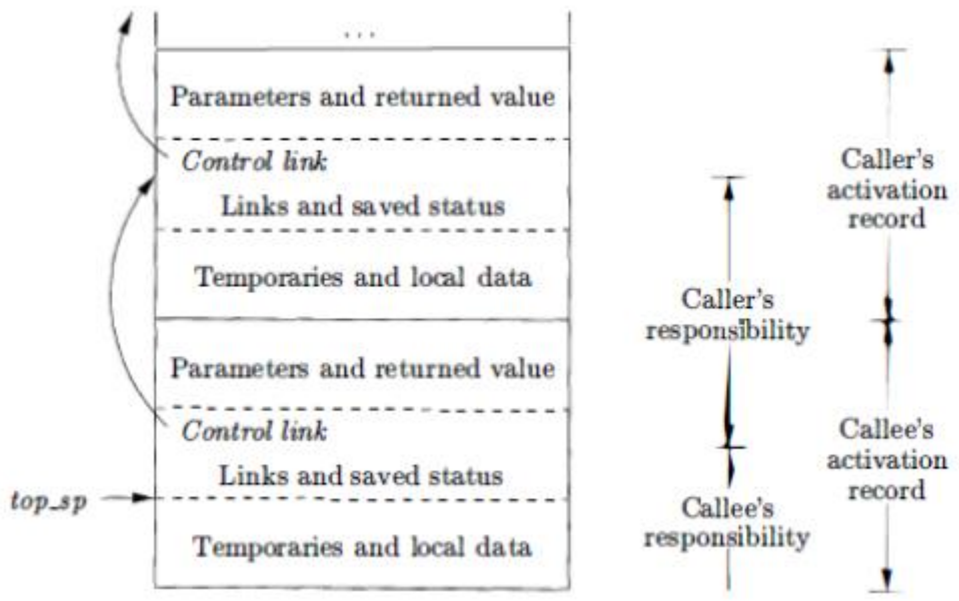
- Procedure calls and returns are usually managed by a run-time stack called the control stack.
- Each live activation has an activation record (sometimes called a frame) on the control stack, with the root of the activation tree at the bottom, and the entire sequence of activation records on the stack corresponding to the path in the activation tree to the activation where control currently resides.
- The latter activation has its record at the top of the stack.



1. Temporary values, such as those arising from the evaluation of expressions, in cases where those temporaries cannot be held in registers.
2. Local data belonging to the procedure whose activation record this is.
3. A saved machine status, with information about the state of the machine just before the call to the procedure.
4. An “access link" may be needed to locate data needed by the called procedure but found elsewhere, e.g., in another activation record.
5. A control link, pointing to the activation record of the caller.
6. Space for the return value of the called function, if any. Again, not all called procedures return a value, and if one does, we may prefer to place that value in a register for efficiency.
7. The actual parameters used by the calling procedure. Commonly, these Values are not placed in the activation record but rather in registers, when possible, for greater efficiency. However, we show a space for them to be completely general.

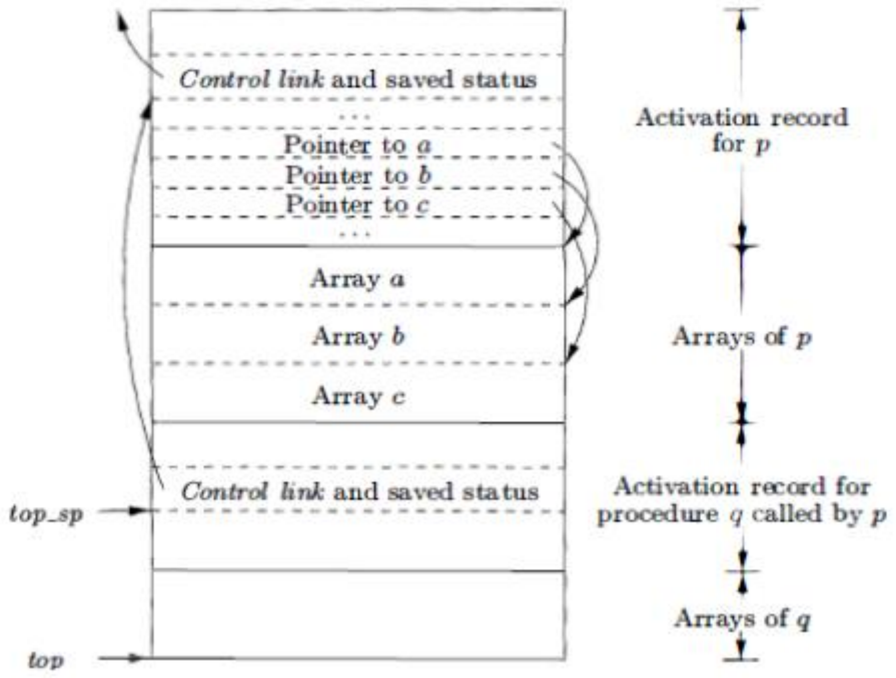
Calling Sequences

- Procedure calls are implemented by what are known as calling sequences, which consists of code that allocates an activation record on the stack and enters information into its fields.
- A return sequence is similar code to restore the state of the machine so the calling procedure can continue its execution after the call.


...
Parameters and returned value
Control link
Links and saved status
Temporaries and local data
Parameters and returned value
Control link
Links and saved status
Temporaries and local data
top_sp
Caller's activation record
Caller's responsibility
Callee's activation record
Callee's responsibility

Variable Length Data on the stack

- The run-time memory-management system must deal with objects whose size are not known at compile time. In modern languages, objects whose size cannot be determined at compile time are allocated space in the heap. However, it is also possible to allocate objects, arrays, or other structures of unknown size on the stack.
- The reason to prefer placing objects on the stack if possible is that we avoid the expense of garbage collecting their space.


Control link and saved status
...
Pointer to a
Pointer to b
Pointer to c
...
Array a
Array b
Array c
Control link and saved status
top_sp
top
Activation record for p
Arrays of p
Activation record for procedure q called by p
Arrays of q