CS 403: Runtime Environments

Stefan D. Bruda

Winter 2015

Runtime Support

- Earlier programming languages had only static, global variables
- Throughout the history of programs many goodies have been added, including local variables (and stack allocation), dynamic memory (and heap allocation), objects, etc.
- This is a brief survey of the runtime support for the features of modern programming languages

Data Representation

- Simple variables are simply represented by sufficiently large memory locations to hold them (e.g., 1 or 3 bytes for characters; 2, 4, or 8 bytes for integers, etc.)
  - Consistent convention for representing values also needed (e.g., binary form with 1's or 2'c complement for integers, the IEEE standard for floating point numbers, etc.)
  - Typically storage of simple variables mimic the capabilities of the underlying machine (if any)
- Pointers are stored as unsigned integers
- One-dimensional arrays are stored as a contiguous block of elements, with location of elements determined using pointer arithmetic
- Multi-dimensional arrays can be stored using two approaches:
  1. Contiguous block for all data, either in row-major (C, Pascal) or column-major (Fortran) order
     - Location of elements determined using pointer arithmetic
  2. Array of arrays, in which one array contains pointers to other arrays (DECAF)
     - Location of elements determined by dereferencing two pointers

Data Representation (cont'd)

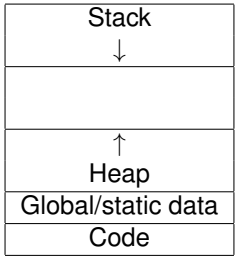
- Structs are laid out by allocating the fields sequentially in a contiguous block
  - Padding is also required on many machines to ensure that all fields start on an aligned boundary
- Objects are stored pretty much like structs
  - Methods could be stored inside objects, but only when they are private and final (Java), or not virtual (C++)
  - Otherwise dynamic dispatch is needed at run time to determine which method to call
  - This is supported by using a hidden extra pointer within each object referencing a shared, class-wide list of methods (vtable)

Object Representation

- First OO complication: inheritance
  - The new fields of an object of type *D* which inherits from *B* are stored at the end (after *B*'s fields)
  - This way a reference of type *B* can also be used to point to objects of type *D*
- Second OO complication: a method must know which object it belongs to
  - Pass the pointer this as an implicit first parameter
- Third OO complication: dynamic dispatch = calling a function at runtime based on the dynamic type of an object rather than its static type
  - A virtual function table (or vtable) is an array of pointers to method implementations
  - Each vtable defined at class level, while each object holds a pointer to the respective vtable
  - When we want to invoke a function we look it up in the vtable and call what we find there
  - Each class defines a vtable; a derived class starts with all the pointers in the vtable pointing to the methods in the base class, and then overwrites pointers as new methods are defined and replace the old ones

Storage Classes

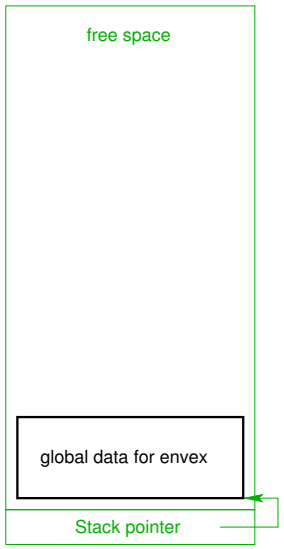
- A variable can be global, static, local, or dynamic
  - Global and static variables are usually stored in a separate segment of the executable
  - Local variables are stored on a stack
  - Dynamic variables are stored on a heap
    - Manual allocation
    - Either manual deallocation (`delete`) or automatic deallocation (garbage collection)
- Typical address space of a program:



Runtime Stack

- The runtime stack holds one stack frame (or activation record) for each active function
- Each frame contains the following information:
  - Frame pointer of dynamic link: pointer to the previous stack frame (so that we can reset the top upon function return)
  - Static link: holds a link to the function in which the current function was declared
    - Only for languages that allow the definition of functions inside functions
  - Return address
  - Arguments passed to the function and local variables
- A stack pointer points to the topmost (active) stack frame
- Also a valid approach to scopes

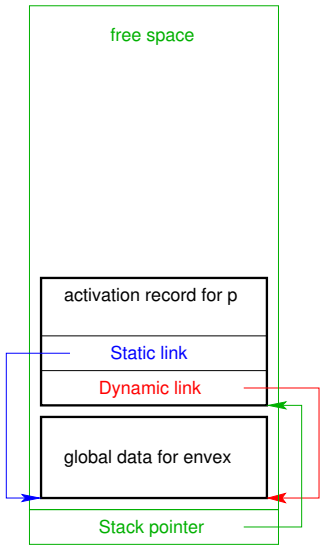
Before `main` calls `p`:



Runtime Stack

- The runtime stack holds one stack frame (or activation record) for each active function
- Each frame contains the following information:
  - Frame pointer of dynamic link: pointer to the previous stack frame (so that we can reset the top upon function return)
  - Static link: holds a link to the function in which the current function was declared
    - Only for languages that allow the definition of functions inside functions
  - Return address
  - Arguments passed to the function and local variables
- A stack pointer points to the topmost (active) stack frame
- Also a valid approach to scopes

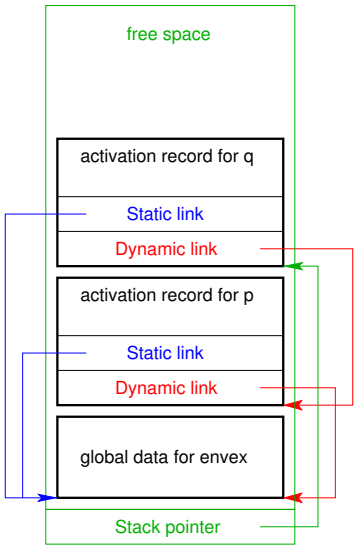
`p` launches:



Runtime Stack

- The runtime stack holds one stack frame (or activation record) for each active function
- Each frame contains the following information:
  - Frame pointer of dynamic link: pointer to the previous stack frame (so that we can reset the top upon function return)
  - Static link: holds a link to the function in which the current function was declared
    - Only for languages that allow the definition of functions inside functions
  - Return address
  - Arguments passed to the function and local variables
- A stack pointer points to the topmost (active) stack frame
- Also a valid approach to scopes

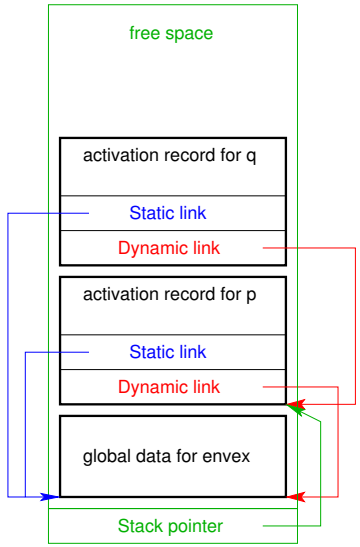
q launches:



Runtime Stack

- The runtime stack holds one stack frame (or activation record) for each active function
- Each frame contains the following information:
  - Frame pointer of dynamic link: pointer to the previous stack frame (so that we can reset the top upon function return)
  - Static link: holds a link to the function in which the current function was declared
    - Only for languages that allow the definition of functions inside functions
  - Return address
  - Arguments passed to the function and local variables
- A stack pointer points to the topmost (active) stack frame
- Also a valid approach to scopes

q returns:



Runtime Stack

- The runtime stack holds one stack frame (or activation record) for each active function
- Each frame contains the following information:
  - Frame pointer of dynamic link: pointer to the previous stack frame (so that we can reset the top upon function return)
  - Static link: holds a link to the function in which the current function was declared
    - Only for languages that allow the definition of functions inside functions
  - Return address
  - Arguments passed to the function and local variables
- A stack pointer points to the topmost (active) stack frame
- Also a valid approach to scopes

p returns: