



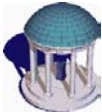
The University of North Carolina at Chapel Hill

COMP 144 Programming Language Concepts
Spring 2003

Control Abstraction

Stotts, Hernandez-Campos

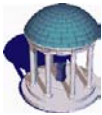
1



Abstraction

- Programming languages support the binding of names with potentially complex program fragments that can be used through an interface
 - Programmers only need to know about the purpose of the fragment rather than its implementation \Rightarrow **Abstraction**
- A **control abstraction** performs a well-defined operation
 - Subroutines
- A **data abstraction** represents information
 - Data structures
 - Most data structures include some number of control abstractions

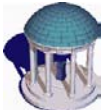
2



Subroutines

- Execute an operation on behalf of a calling program unit
- Subroutines can be parameterized
 - The parameters in the definition of the function are known as *formal parameters*
 - The parameters passed in the subroutine call are known as *actual parameters* or *arguments*
 - At the time of the call, actual parameters are mapped to formal parameters
- *Functions* are subroutines that return a value, while *procedures* are subroutines that do not return a value

3

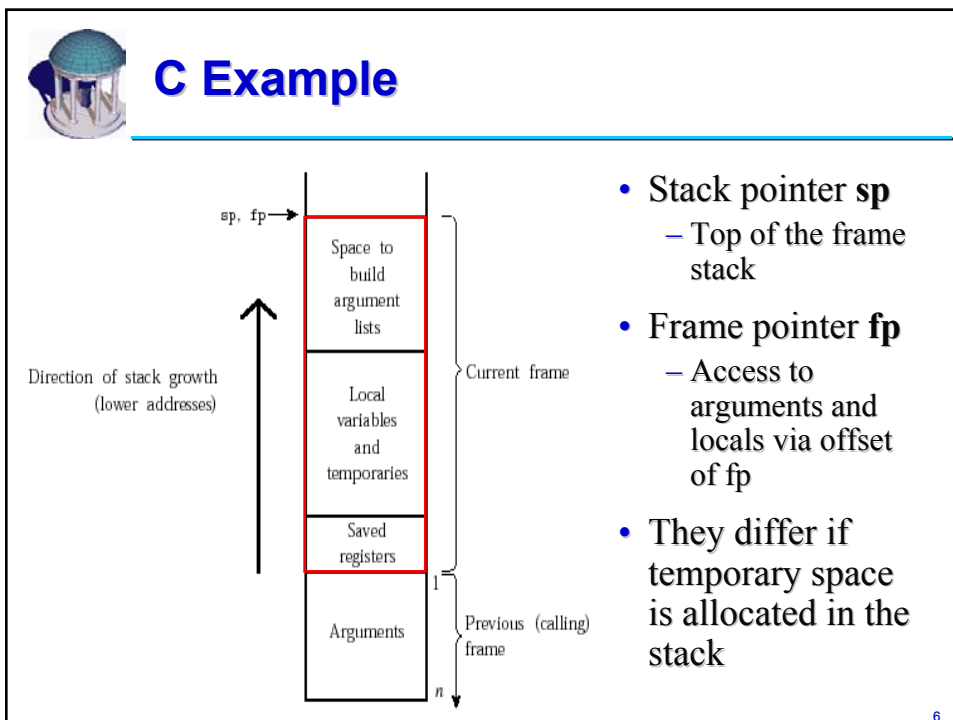
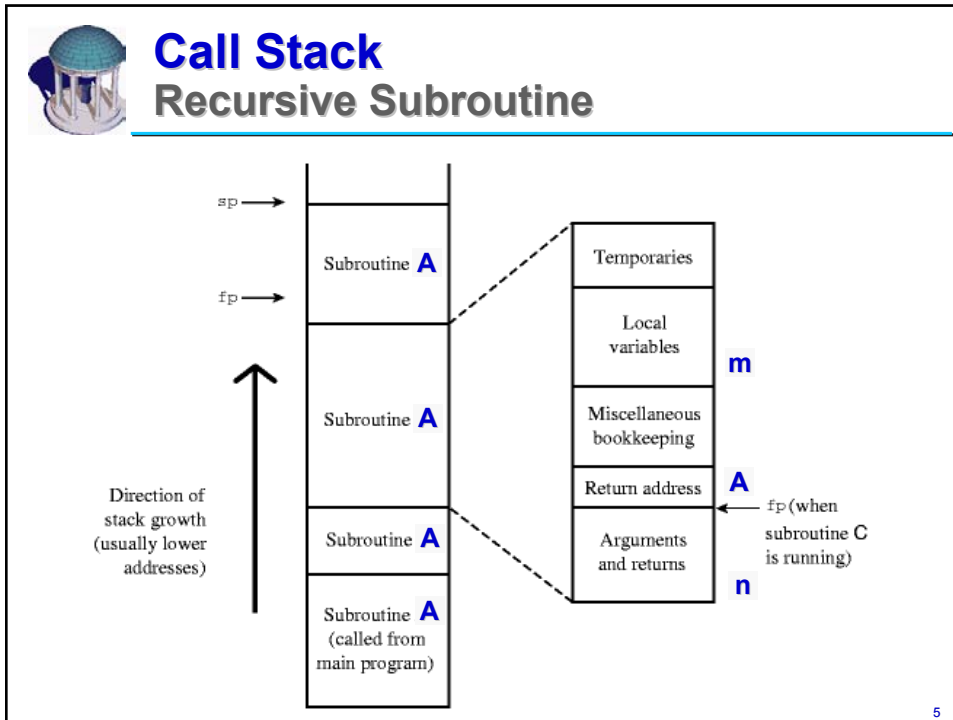



Subroutine Frames

```
1001: A(3) → Actual Parameters
...
2001: int A(int n) { → Formal Parameters
        int m = n * n;
        return m + A(n-1);
    }
```

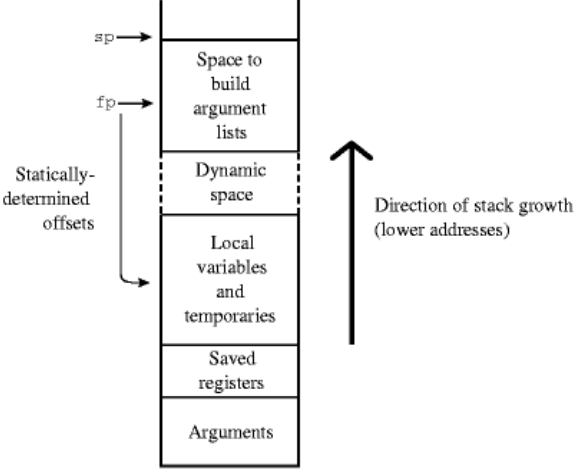
- Each subroutines requires a *subroutine frame* (a.k.a *activation record*) to keep track of
 - Arguments and return values
 - Local variables and temporaries
 - Bookkeeping information
- When a subroutine returns, its frame is removed

4






C Example



The diagram illustrates the stack layout. It is a vertical rectangle divided into several sections. From top to bottom, the sections are: 'Space to build argument lists', 'Dynamic space' (indicated by a dashed line), 'Local variables and temporaries', 'Saved registers', and 'Arguments'. On the left side, two arrows point to the top of the stack: 'sp' (stack pointer) at the very top, and 'fp' (frame pointer) at the top of the 'Space to build argument lists' section. A bracket labeled 'Statically-determined offsets' spans from the 'fp' level down to the 'Arguments' section. On the right side, a vertical arrow points upwards, labeled 'Direction of stack growth (lower addresses)'. A small number '7' is in the bottom right corner.

- Stack pointer **sp**
 - Top of the frame stack
- Frame pointer **fp**
 - Access to arguments and locals via offset of fp
- **They differ if temporary space is allocated in the stack**



Stack Maintenance

- Calling sequence (by caller) and prologue (by callee) are executed in the way into the subroutine:
 - Pass parameters
 - Save return address
 - Change program counter
 - Change stack pointer to allocate stack space
 - Save registers (including frame pointer)
 - Change frame pointer to new frame
 - Initialization code
- Separation of tasks?
 - As much as possible in the callee (only once in the program)

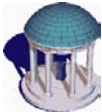
A small number '8' is in the bottom right corner.



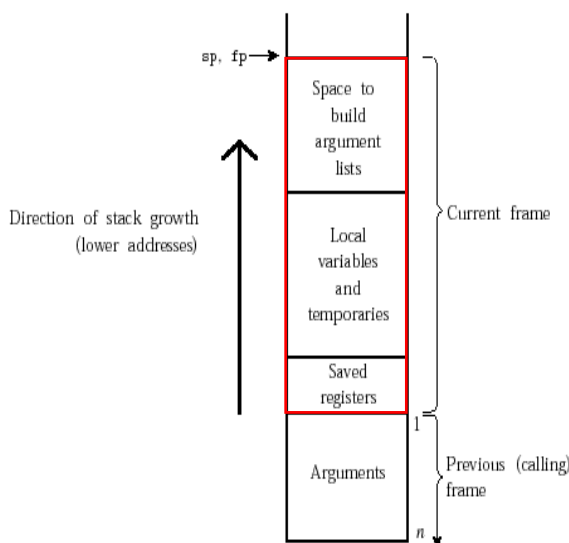
Stack Maintenance

- Epilogue (by callee) is executed in the way out of the subroutine:
 - Pass return value
 - Execute finalization code
 - Deallocate stack frame (restore stack pointer)
 - Restore registers

9




C Example



Calling sequence by the caller:

1. Caller saves registers in local variables and temporaries space
2. 4 scalar arguments in registers
3. Rest of the arguments at the top of the current frame
4. Return address in register **ra** and jump to target address

10



C Example

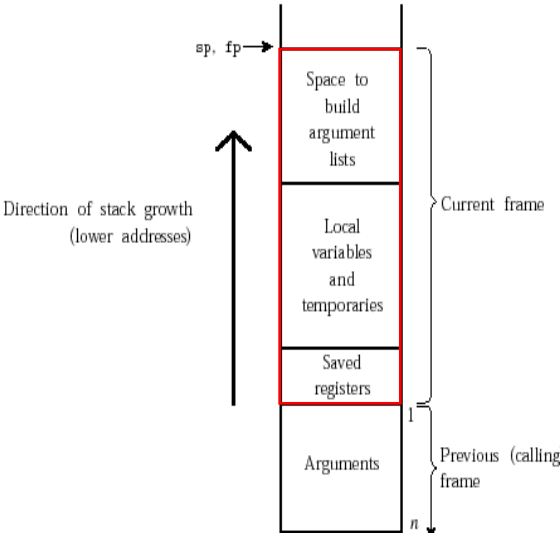



Diagram illustrating the stack layout during the prologue of a function call. The stack grows downwards (higher addresses at the top, lower addresses at the bottom). The stack is divided into two main sections: the Current frame and the Previous (calling) frame. The Current frame contains, from top to bottom: Space to build argument lists, Local variables and temporaries, and Saved registers. The Previous (calling) frame contains Arguments. The stack pointer (sp) and frame pointer (fp) are shown pointing to the top of the Current frame. The stack grows from address n at the bottom to address i at the top.

Prologue by the callee:

1. Subtract the frame size from the sp
2. Save registers at the beginning of the new frame using sp as the base for displacement
3. Copy the sp into the fp

11



C Example

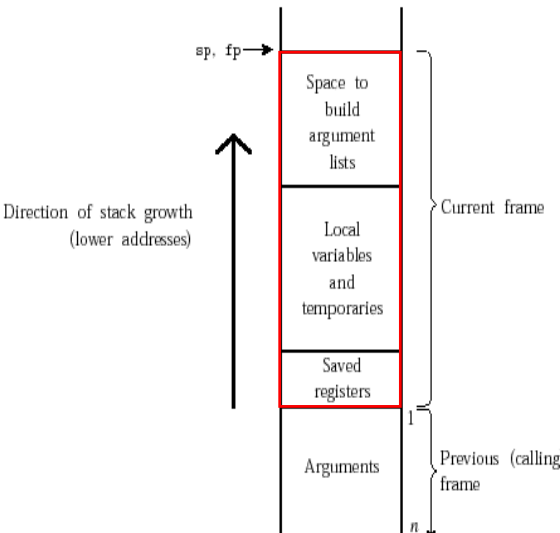
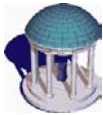


Diagram illustrating the stack layout during the epilogue of a function call. The stack grows downwards (higher addresses at the top, lower addresses at the bottom). The stack is divided into two main sections: the Current frame and the Previous (calling) frame. The Current frame contains, from top to bottom: Space to build argument lists, Local variables and temporaries, and Saved registers. The Previous (calling) frame contains Arguments. The stack pointer (sp) and frame pointer (fp) are shown pointing to the top of the Current frame. The stack grows from address n at the bottom to address i at the top.

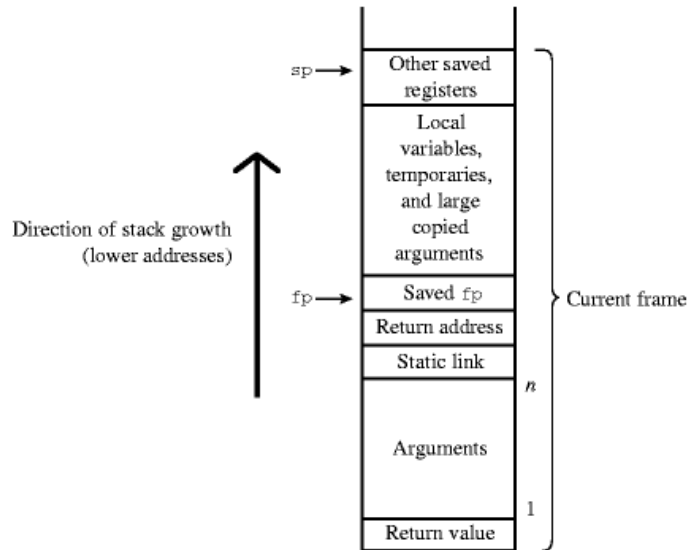
Callee epilogue:

1. Set the function return value
2. Copy fp to sp to deallocate any dynamically allocated space
3. Restore registers including ra (based on sp)
4. Add frame size to sp
5. Return (jump to ra)

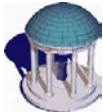
12



Pascal Example



13



Parameter Passing

- Pass-by-value
 - Input parameter
- Pass-by-result
 - Output parameter
- Pass-by-value-result
 - Input/output parameter
- Pass-by-reference
 - Input/Output parameter, no copy
- Pass-by-name
 - Textual substitution

14



Closure

- A Closure is a subroutines that maintains all its referencing environment
 - This mechanism is also known as *deep binding* or *deep access*
- This is significant when subroutines are passed as arguments

15



Reading Assignment

- Read Scott
 - Sect. 8.1-3

16