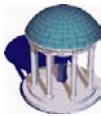


The University of North Carolina at Chapel Hill

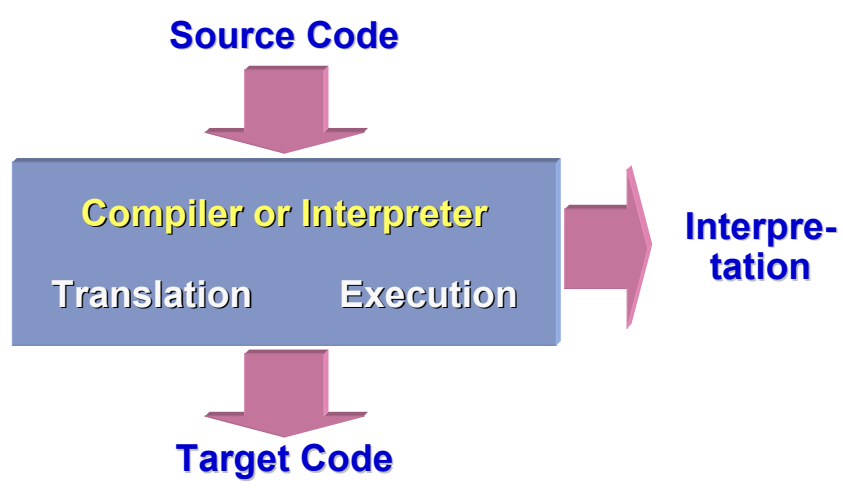
COMP 144 Programming Language Concepts
Spring 2003

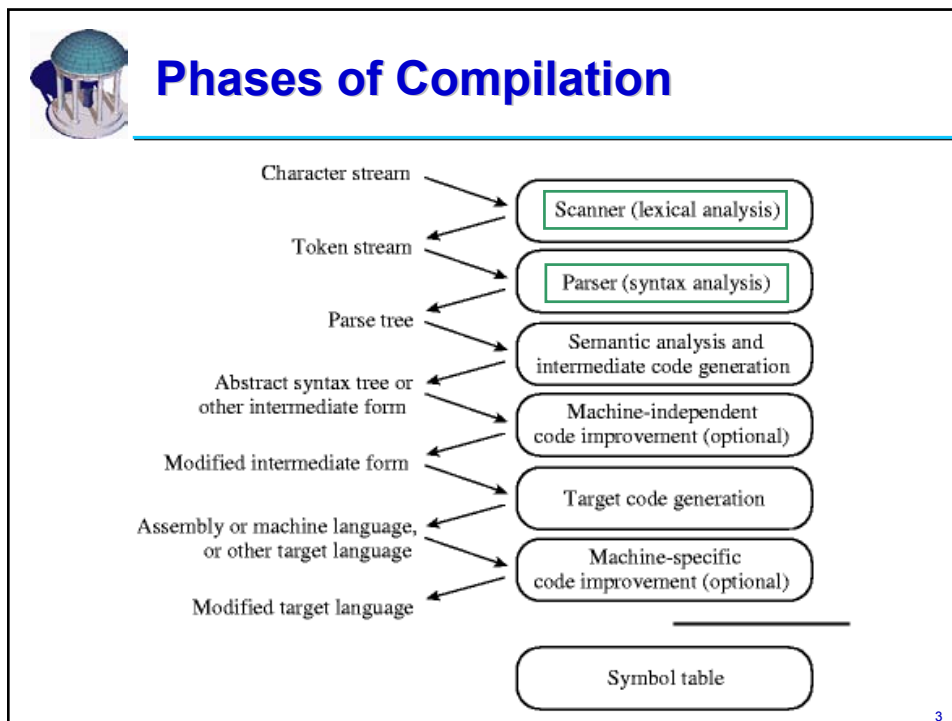
Binding Time and Storage Allocation

Stotts, Hernandez-Campos



Review: Compilation/Interpretation



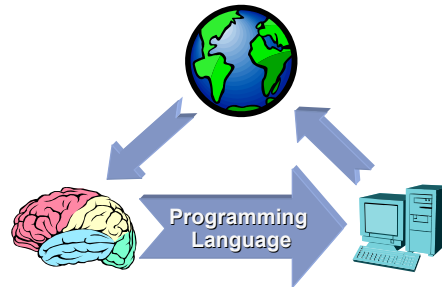


-
- ## High-Level Programming Languages
- Two main goals:
 - *Machine independence*
 - *Ease of programming*
 - High-level programming language are independent of any particular instruction set
 - But compilation to assembly code requires a target instruction set
 - There is a trade-off between machine independence and efficiency
 - » *E.g. Java vs. C*
- 4

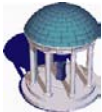


Ease of Programming

- The driving problem in programming language design
- The rest of this class
 - Names
 - Control Flow
 - Types
 - Subroutines
 - Object Orientation
 - Concurrency
 - Declarative Programming



5



Naming

- **Naming** is the process by which the programmer associates a name with a potentially complicated program fragment
 - The goal is to hide complexity
 - Programming languages use name to designate variables, types, classes, methods, operators,...
- Naming provides *abstraction*
 - *E.g.* Mathematics is all about the formal notation (i.e. naming) that lets us explore more and more abstract concepts

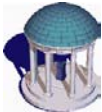
6



Control and Data Abstraction

- **Control abstraction** allows the programmer to hide an arbitrarily complicated code behind a simple interface
 - Subroutines
 - Classes
- **Data Abstraction** allows the programmer to hide data representation details behind abstract operations
 - ADTs
 - Classes

7



Binding Time

- A **binding** is an association between two things
 - E.g. Name of an object and the object
- **Binding time** is the time at which a binding is created
 - Language design time
 - Language implementation
 - Program writing time
 - Compile Time
 - Link Time
 - Load Time
 - Run Time

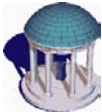
8



Binding Time Impact

- Binding times have a **crucial** impact in programming languages
 - They are a fundamental design decision
- In general, early binding is associated with greater efficiency
 - *E.g.* C string vs. Java's StringBuffer
- In general, late binding is associated with greater flexibility
 - *E.g.* Class method vs. Subroutine
- Compiled languages tend to bind names earlier than interpreted languages

9



Object Lifetime

- Events in the life of an object
 - Creation
 - Destruction } **Object Lifetime**
- Events in the life of a binding
 - Creation
 - Destruction
 - » A binding to an object that no longer exists is called a *dangling reference*
 - Deactivation and Reactivation

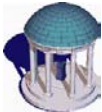
10



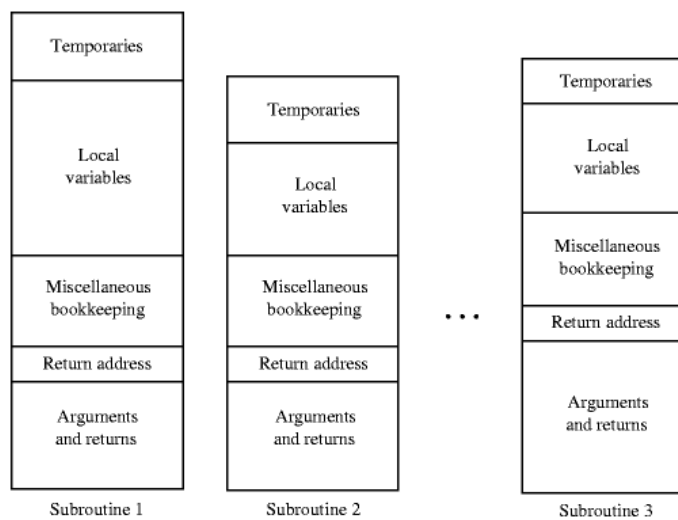
Storage Allocation Mechanisms

- In **static allocation**, objects are given an absolute address that is retained throughout the program's execution
 - E.g. Global variables, Non-recursive Subroutine Parameters

11



Static Allocation



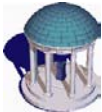
12



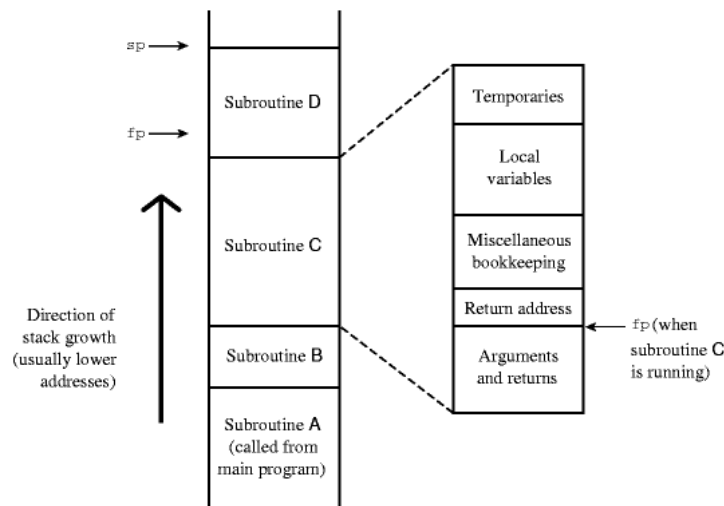
Storage Allocation Mechanisms

- In **static allocation**, (static) objects are given an absolute address that is retained throughout the program's execution
 - E.g. Global variables, Non-recursive Subroutine Parameters
- In **stack-based allocation**, (stack) objects are allocated in last-in, first-out data structure, a *stack*.
 - E.g. Recursive subroutine parameters

13



Stack-based Allocation



14



Calling Sequence

- On procedure (method) *call* and *return* compiler generates code that gets executed to manage the runtime stack
- **Setup** at call to procedure *doFoo(a,b)*
- **Prologue** before *doFoo* code itself
- **Epilogue** at end of *doFoo* code
- **“Teardown”** in calling code right after call

15



Calling Sequence

- **Setup** at call to procedure *doFoo(a,b)*
 - Move *sp* to allocate a new stack frame
 - Copy args *a,b* into frame
 - Copy return address into frame
 - Set *fp* to point to new frame
 - maintain static chain or display
- **Prolog** before procedure code itself
 - Copy registers into local slots
 - Object initialization

16



Calling Sequence

- **Epilog** at end of procedure code
 - Placing return value into slot in frame
 - Restore registers
 - Restore PC to return address
- **“teardown”** in calling code right after call
 - Move *sp* (deallocate frame)
 - Move *fp* using dynamic chain
 - Maybe move return values (if in registers)

17



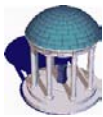
Calling Sequence: Perl

- In Perl you have to manage some of the calling sequence yourself

```
&doFoo($a, $b, \@ar5);
```

```
sub doFoo {  
  my ($arg1, $arg2, $ref1) = @_ ; # aha!  
  # not quite the same  
  $_[0] = 4;  
  $_[1] = $$@[2][7]; # etc is ok  
}
```

18



Storage Allocation Mechanisms

- In **static allocation**, (static) objects are given an absolute address that is retained throughout the program's execution
 - E.g. Global variables, Non-recursive Subroutine Parameters
- In **stack-based allocation**, (stack) objects are allocated in last-in, first-out data structure, a *stack*.
 - E.g. Subroutine parameters
- In **heap-based allocation**, (heap) objects may be allocated and deallocated at arbitrary times
 - E.g. objects created with C++ `new` and `delete`

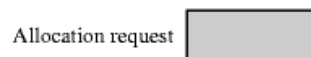
19



Heap-based Allocation

The **heap** is a region of storage in which subblock can be allocated and deallocated

- This not the *heap data structure*



Oops... no fit...

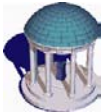
20



Heap Space Management

- In general, the heap is allocated sequentially
- This creates space **fragmentation** problems
 - **Internal fragmentation**
 - » If size of object to allocated is larger than the size of the available heap
 - » Come from fixed size heap blocks
 - **External fragmentation**
 - » If size of object to allocated is not larger than the size of the available heap, but the available space in the heap is scattered through heap in such a way that no contiguous space fits
 - » Comes from variable size blocks

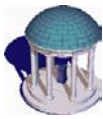
21



Heap Space Management

- Some languages (C) require (allow) **explicit heap management** ...
allocation and de-allocation of dynamic storage by the programmer
 - *malloc, return*
- **Easy to forget return**
 - Heap fills with objects no longer in use (dangling refs)
 - Memory leaks... heap gets exhausted

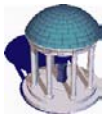
22



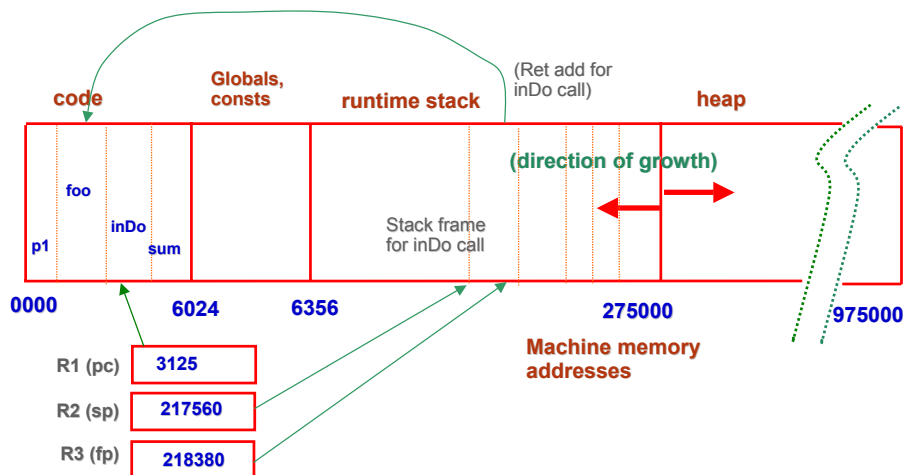
Heap Space Management

- Some languages (Java) **manage the heap for you**
 - new() object allocated on heap
 - when done, the object storage is reclaimed
- Automatic de-allocation after an object has no bindings/references is called **garbage collection**
 - E.g. Java
 - some runtime efficiency hit
 - no memory leaks

23



Sample Memory Layout



24



Reading Assignment

- Scott's chapter 3
 - Section 3.1
 - Section 3.2