Names, Scopes, Bindings

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

Limitations of CFG

Grammar for a simple language:

```
Prog ::= Decl* Stmt*
Decl ::= int ID; | bool ID;
Stmt ::= ID = Exp
```

Need Context Sensitive Information

```
int x;
y = 0
```

y defined?

```
bool b;
b = 0
```

b number?

We need to add ***meanings*** to symbols when go beyond syntactic analysis (binding: symbol to object)

Name

A mnemonic string in high-level languages

Identifiers in most languages

An abstraction of low-level representation, such as memory address and register

Questions about Names

How are names introduced?

```
int X;
```

```
(define x 3)
```

When are names given meanings?

```
void foo () {
  int count = 0;
  … }
```

Can names be redefined?

```
(define (if) 1)
```

How are names resolved? When?

Variables

A ***binding*** of a name to a memory address

A variable usually has:

name, address, type, value, lifetime, scope

l-value vs. r-value

```
x = y + 1;
```

Names in Programs

Scope: visibility of names

Storage: memory space associated with names

Lifetime: the time interval a variable is allocated with memory

Binding: a mapping between a name and its property

Bindings

A mapping bet. a name and its property

When are they made? In C:

- Language design time: bind operator symbols
- Language impl. time: a type int is bound to a range of possible values on an architecture
- Compile time: bind a variable to a particular data type
- Load time: bind a global variable to a memory cell
- Run time: bind a local variable when a function is active

Bindings

Try this in Scheme

```
(define + *)
(+ 3 4)
```

```
(define (if e t f) f)
(if #t #t #f)
```

When are operators/keywords bound in Scheme?

Objects

Run-time representation of a name

Key events of objects:

- Creation of objects
- References to variables (which use bindings)
- (temporary) deactivation of bindings
- Reactivation of bindings
- Destruction of bindings
- Destruction of objects

Lifetime

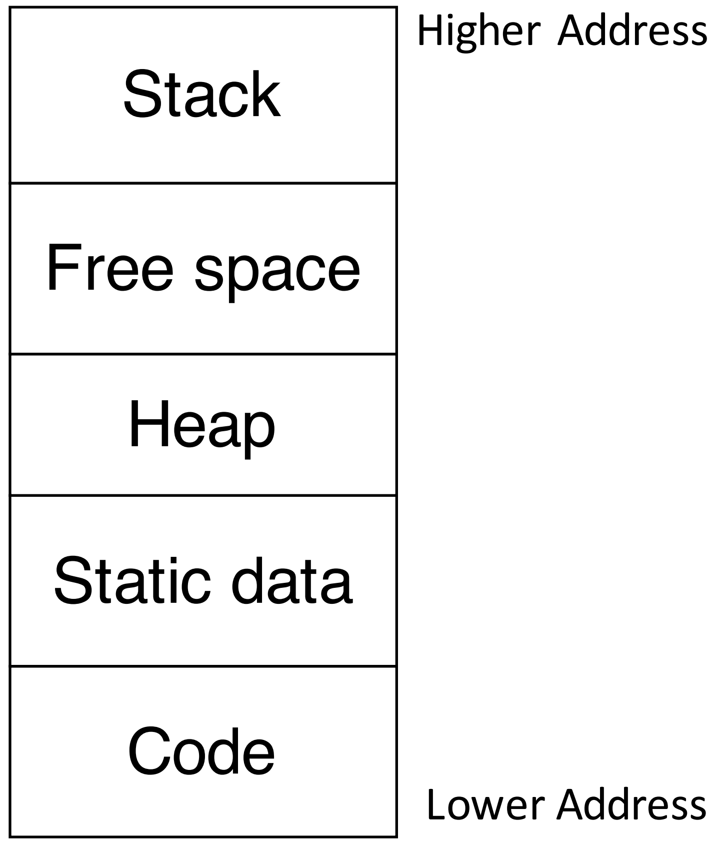
Binding: name to object

- Lifetime of name
- Lifetime of object
- Dependent or independent?

Scope of binding

- Static or dynamic

Typical Memory Layout



Storage Allocation Mechanism and Object Lifetime

Static allocation: bound to memory cells before execution begins and remains bound to the same memory cell throughout execution

Stack allocation: storage bindings are created for variables when their declaration statements are elaborated

Heap allocation: Nameless memory cells that are allocated and deallocated by explicit directives “run-time instructions”, specified by the programmer (either explicitly or implicitly)

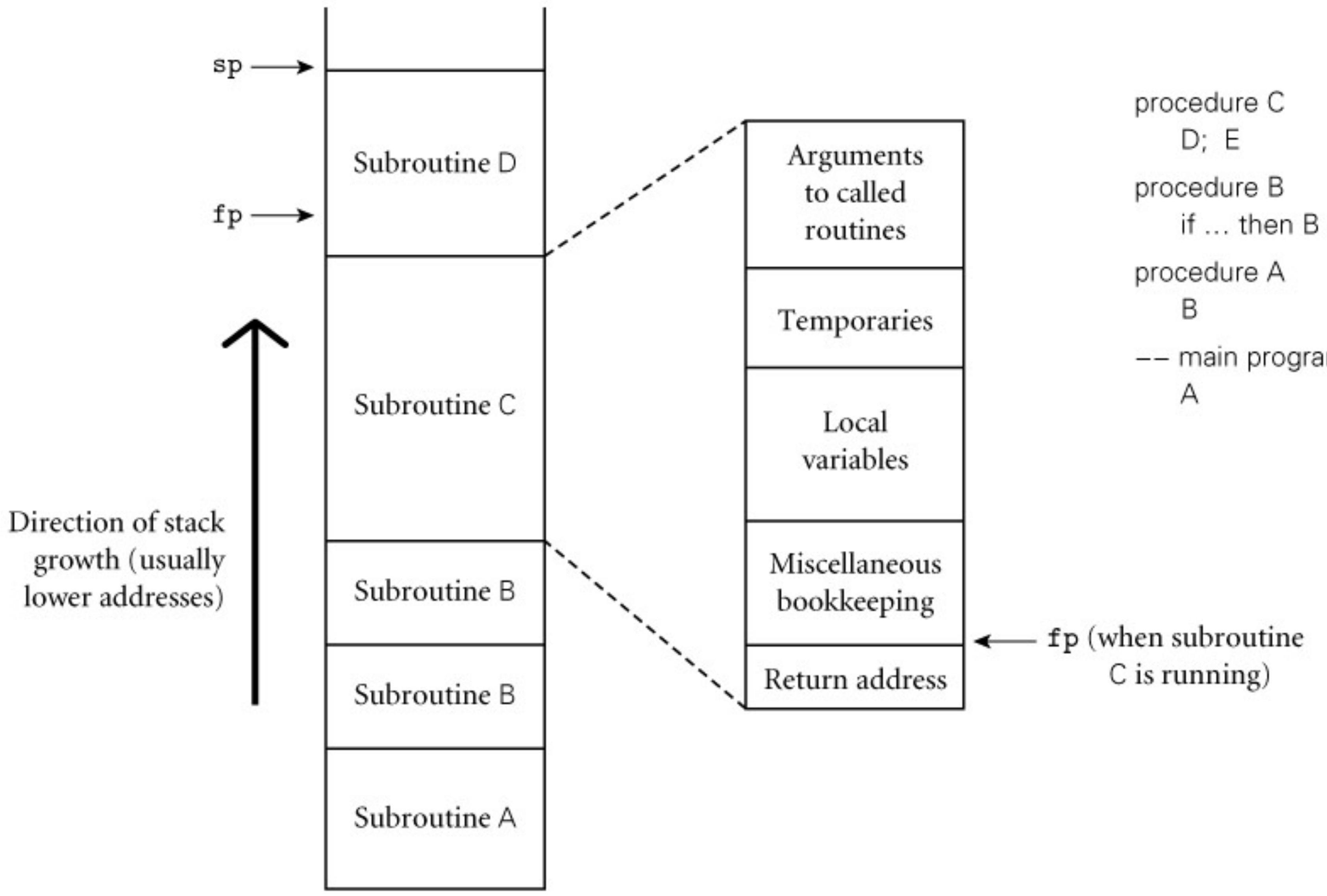
Static Allocation

Lifetime of Objects: throughout execution

E.g., global variables, constants

Local variables in FORTRAN 77 are statically allocated. Consequence?

Stack Allocation – Activation Record


sp
fp
Subroutine D
Subroutine C
Subroutine B
Subroutine B
Subroutine A
Direction of stack growth (usually lower addresses)
Arguments to called routines
Temporaries
Local variables
Miscellaneous bookkeeping
Return address
fp (when subroutine C is running)

```
procedure C
    D;  E
procedure B
    if ... then B else C
procedure A
    B
-- main program
    A
```

Lifetime of Local Variables

When does the lifetime of each variable begin & end?

How do these lifetimes relate to each other?

```
int main () {
  int x;
  p(x);
  q(x,x);
}
```

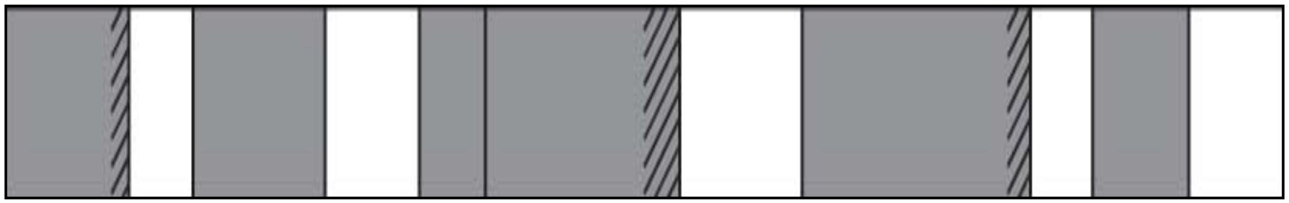
```
int p(int p1) {
  int px;
  q(p1,px);
}
```

```
int q(int q1,q2) {
  int qx;
  ...
}
```

Heap Allocation

Allocation Request (size)

Free request



Lifetime of User-Requested Storage

When does the lifetime of dynamic storage begin & end?

How do these lifetimes relate to each other?

```
int main () {
  int x;
  p();
  q();
}
```

```
int p() {
  ...
  x = new String[10];
  ...
}
```

```
int q() {
  ...
  delete x
  ...
}
```

Garbage Collection

Explicit or implicit heap deallocation

- Explicit (by programmer): C, C++, Pascal
- Implicit (by garbage collector): Java, C#, Scheme

Pros and Cons?