Direction of stack growth (usually lower addresses)

- Subroutine A
- Subroutine B
- Subroutine C
- Subroutine D

Arguments to called routines
- Temporaries
- Local variables
- Miscellaneous bookkeeping
- Return address

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

sp
fp
fp (when subroutine C is running)
Heap

Allocation request
Place into *s a new name beginning with the letter 'L' and continuing with the ASCII representation of a unique integer. Parameter s is assumed to point to space large enough to hold any such name; for the short ints used here, 7 characters suffice.

```c
void label_name (char *s) {
    static short int n;            /* C guarantees that static locals are initialized to zero */
    sprintf (s, "L%d\0", ++n);    /* "printf" formatted output to s */
}
```
procedure P1(A1)
  var X    -- local to P1
  ...
procedure P2(A2)
  ...
  procedure P3(A3)
  ...
  begin
  ...
  -- body of P3
  end
  ...
  begin
  ...
  -- body of P2
  end
  ...
procedure P4(A4)
  ...
  function F1(A5)
    var X  -- local to F1
    ...
  begin
  ...
  -- body of F1
  end
  ...
  begin
  ...
  -- body of P4
  end
  ...
begin
  ...
-- body of P1
end
#include <time.h>

namespace rand_mod {

    unsigned int seed = time(0);   // initialize from current time of day
    const unsigned int a = 48271;
    const unsigned int m = 0x7fffffff;

    void set_seed(unsigned int s) {
        seed = s;
    }

    unsigned int rand_int() {
        return seed = (a * seed) % m;
    }
}

#include <time.h>

namespace rand_mgr {

    const unsigned int a = 48271;
    const unsigned int m = 0x7fffffff;

    typedef struct {
        unsigned int seed;
    } generator;

    generator* create() {
        generator* g = new generator;
        g->seed = time(0);
        return g;
    }

    void set_seed(generator* g, unsigned int s) {
        g->seed = s;
    }

    unsigned int rand_int(generator* g) {
        return g->seed = (a * g->seed) % m;
    }
}

class rand_gen {
    unsigned int seed = time(0);
    const unsigned int a = 48271;
    const unsigned int m = 0x7fffffffff;

public:
    void set_seed(unsigned int s) {
        seed = s;
    }

    unsigned int rand_int() {
        return seed = (a * seed) % m;
    }
};
1. n : integer  
   -- global declaration

2. procedure first()

3.     n := 1

4. procedure second()

5.     n : integer  
      -- local declaration

6.     first()

7.     n := 2

8.     if read_integer() > 0

9.         second()

10.    else

11.         first()

12.    write_integer(n)
max_score : integer  -- maximum possible score

function scaled_score(raw_score : integer) : real
    return raw_score / max_score * 100

...

procedure foo()
    max_score : real := 0  -- highest percentage seen so far
...

    foreach student in class
        student.percent := scaled_score(student.points)
        if student.percent > max_score
            max_score := student.percent


dec

type month is (jan, feb, mar, apr, may, jun,
             jul, aug, sep, oct, nov, dec);

type print_base is (dec, bin, oct, hex);

mo : month;

pb : print_base;

begin

   mo := dec;       -- the month dec (since mo has type month)

   pb := oct;       -- the print_base oct (since pb has type print_base)

   print(oct);     -- error! insufficient context

   -- to decide which oct is intended
struct complex {
    double real, imaginary;
};

enum base {dec, bin, oct, hex};

int i;
complex x;

void print_num(int n) {
 ...
}
void print_num(int n, base b) {
 ...
}
void print_num(complex c) {
 ...
}

print_num(i);  // uses the first function above
print_num(i, hex);  // uses the second function above
print_num(x);  // uses the third function above
type person = record
    ...
    age : integer
    ...
threshold : integer
people : database

function older_than_threshold(p : person) : boolean
    return p.age ≥ threshold

procedure print_person(p : person)
    --- Call appropriate I/O routines to print record on standard output.
    --- Make use of nonlocal variable line_length to format data in columns.
    ...

procedure print_selected_records(db : database;
    predicate, print_routine : procedure)
    line_length : integer

    if device_type(stdout) = terminal
        line_length := 80
    else  --- Standard output is a file or printer.
        line_length := 132
    foreach record r in db
        --- Iterating over these may actually be
        --- a lot more complicated than a 'for' loop.
        if predicate(r)
            print_routine(r)

--- main program
    ...
threshold := 35
print_selected_records(people, older_than_threshold, print_person)
def A(I, P):
    def B():
        print(I)
    # body of A:
    if I > 1:
        P()
    else:
        A(2, B)

def C():
    pass  # do nothing
A(1, C)  # main program
main program

A

I == 2
P == B

A

I == 1
P == C
main program

\[ \text{plus-x} \]

\[
x = 2
\]

\[ \text{rtn} = \text{anon} \]

\[
anon \quad y = 3
\]

main program
typedef struct list_node {
    void* data;
    struct list_node* next;
} list_node;

list_node* insert(void* d, list_node* L) {
    list_node* t = (list_node*) malloc(sizeof(list_node));
    t->data = d;
    t->next = L;
    return t;
}

list_node* reverse(list_node* L) {
    list_node* rtn = 0;
    while (L) {
        rtn = insert(L->data, rtn);
        L = L->next;
    }
    return rtn;
}

void delete_list(list_node* L) {
    while (L) {
        list_node* t = L;
        L = L->next;
        free(t->data);
        free(t);
    }
}