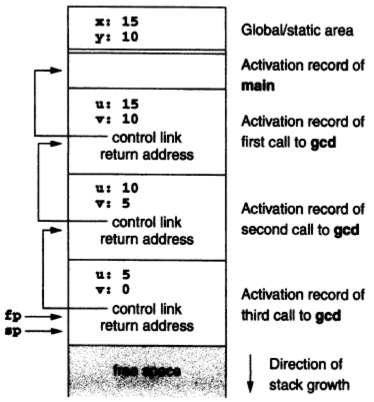
Stack-Based Runtime Environment without Local Procedure(s)

- FP: Frame Pointer – Pointer to the current Activation Record
- CL: Control Link – Pointer to the previous Activation Record (Old FP)
- SP: Stack Pointer

Program:

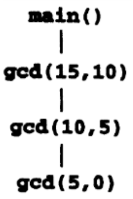
```
#include <stdio.h>

int x,y;

int gcd( int u, int v)
{ if (v == 0) return u;
  else return gcd(v,u % v);
}

main()
{ scanf("%d%d",&x,&y);
  printf("%d\n",gcd(x,y));
  return 0;
}
```

Runtime Environment:



x: 15
y: 10
Global/static area
Activation record of main
u: 15
v: 10
control link
return address
Activation record of first call to gcd
u: 10
v: 5
control link
return address
Activation record of second call to gcd
u: 5
v: 0
control link
return address
Activation record of third call to gcd
fp
sp
free space
Direction of stack growth

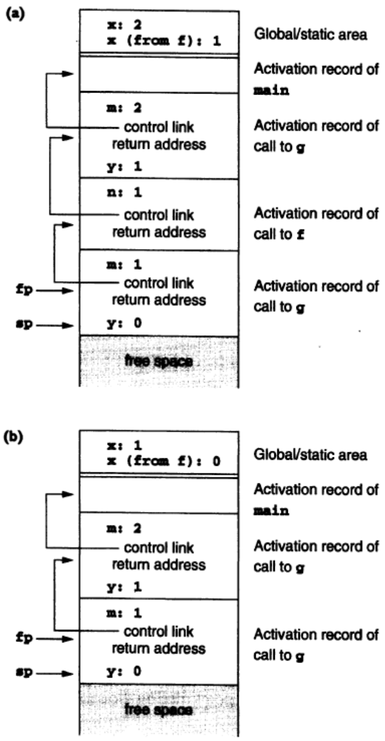
Activation Tree:



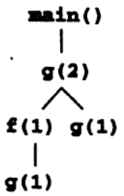
Program:

```
int x = 2;

void g(int); /* prototype */

void f(int n)
{ static int x = 1;
  g(n);
  x--;
}

void g(int m)
{ int y = m-1;
  if (y > 0)
  { f(y);
    x--;
    g(y);
  }
}

main()
{ g(x);
  return 0;
}
```

Runtime Environment:



Activation Tree:



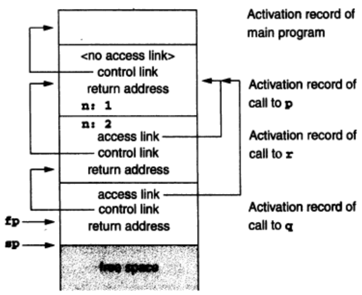
Task: Draw Step-Wise environment for example #1.

Stack-Based Runtime Environment with Local Procedure(s)

Program:

```
program nonLocalRef;

procedure p;
var n: integer;

      procedure q;
      begin
        (* a reference to n is now
           non-local non-global *)
      end; (* q *)


      procedure r(n: integer);
      begin
        q;
      end; (* r *)

begin (* p *)
  n := 1;
  r(2);
end; (* p *)

begin (* main *)
  p;
end.
```

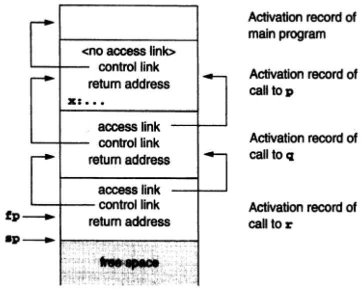
Runtime Environment:



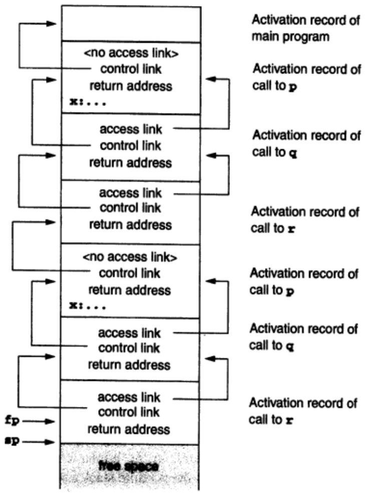
Program:

```
program chain;

procedure p;
var x: integer;

   procedure q;
      procedure r;
      begin
        x := 2;
        ...
        if ... then p;
      end; (* r *)
   begin
     r;
   end; (* q *)

begin
  q;
end; (* p *)

begin (* main *)
  p;
end.
```

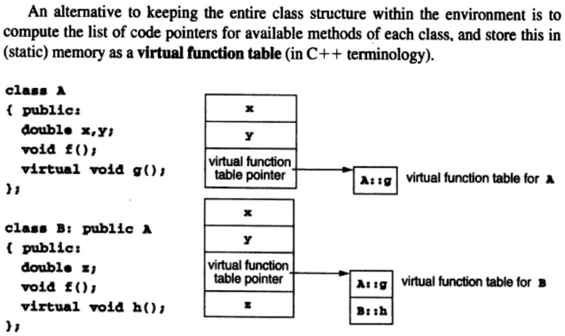
Runtime Environment (1^{st} call to r):



Runtime Environment (2^{nd} call to r):



Dynamic Memory in Object Oriented Languages:



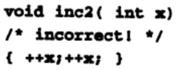
An alternative to keeping the entire class structure within the environment is to compute the list of code pointers for available methods of each class, and store this in (static) memory as a **virtual function table** (in C++ terminology).

```
class A
{ public:
  double x,y;
  void f();
  virtual void g();
};

class B: public A
{ public:
  double z;
  void f();
  virtual void h();
};
```

x
y
virtual function table pointer
A::g
virtual function table for A
x
y
virtual function table pointer
z
A::g
B::h
virtual function table for B

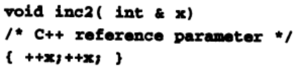
Parameter Passing Mechanisms:

- Pass by Value



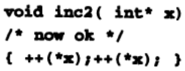
```
void inc2( int x)
/* incorrect! */
{ ++x;++x; }
```

- Pass by reference (Use of Alias)



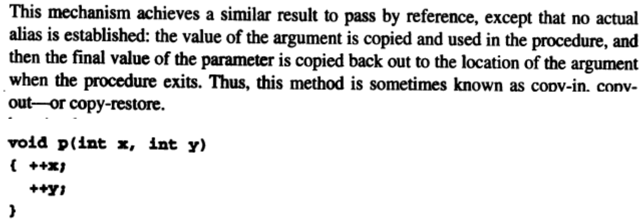
```
void inc2( int & x)
/* C++ reference parameter */
{ ++x;++x; }
```

- Pass by Pointer



```
void inc2( int* x)
/* now ok */
{ ++(*x);++(*x); }
```

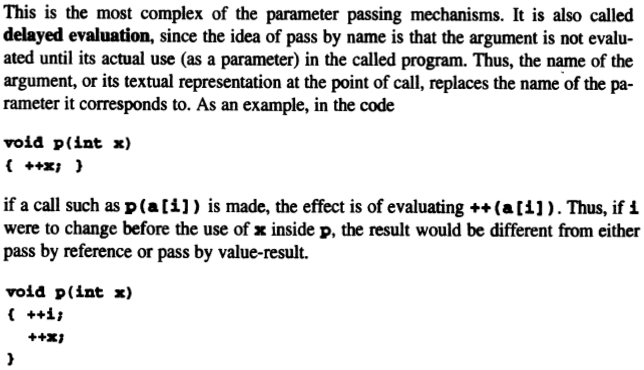
- Pass by Value-result



This mechanism achieves a similar result to pass by reference, except that no actual alias is established: the value of the argument is copied and used in the procedure, and then the final value of the parameter is copied back out to the location of the argument when the procedure exits. Thus, this method is sometimes known as copy-in, copy-out—or copy-restore.

```
void p(int x, int y)
{ ++x;
  ++y;
}
```

- Pass by Name

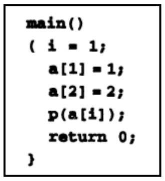


This is the most complex of the parameter passing mechanisms. It is also called **delayed evaluation,** since the idea of pass by name is that the argument is not evaluated until its actual use (as a parameter) in the called program. Thus, the name of the argument, or its textual representation at the point of call, replaces the name of the parameter it corresponds to. As an example, in the code

```
void p(int x)
{ ++x; }
```

if a call such as `p(a[i])` is made, the effect is of evaluating `++(a[i])`. Thus, if `i` were to change before the use of `x` inside `p`, the result would be different from either pass by reference or pass by value-result.

```
void p(int x)
{ ++i;
  ++x;
}
```



```
main()
{ i = 1;
  a[1] = 1;
  a[2] = 2;
  p(a[i]);
  return 0;
}
```