Compiler

--- Run-time Environment

Zhang Zhizheng
seu_zzz@seu.edu.cn
School of Computer Science and Engineering,
Software College
Southeast University

Overview

1.Program vs program execution

- A program consists of several procedures (functions)
- An execution of a program is called as a process
- An execution of a program would cause the activation of the related procedures
- A name (e.g, a variable name)in a procedure would be related to different data objects

Notes:

An activation may manipulate data objects allocated for its use.

Overview II

2. Allocation and de-allocation of data objects

- Managed by the run-time support package
- Allocation of a data object is just to allocate storage space to the data object relating to the name in the procedure

Notes:

The representation of a data object at run time is determined by its type.

Overview III

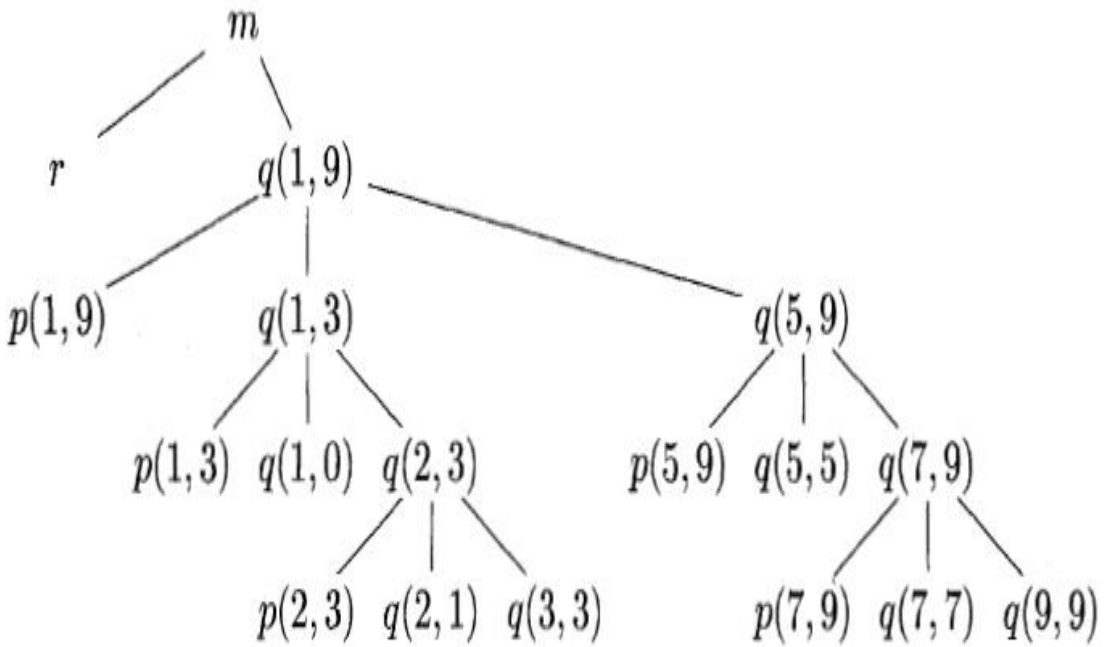
3.Activation Trees

A tree to depict the control flow enters and leaves activation of a procedure

```
int a[11];
void readArray() { /* Reads 9 integers into a[1], ..., a[9]. */
    int i;
    ...
}
int partition(int m, int n) {
    /* Picks a separator value v, and partitions a[m..n] so that
       a[m..p-1] are less than v, a[p] = v, and a[p+1..n] are
       equal to or greater than v. Returns p. */
    ...
}
void quicksort(int m, int n) {
    int i;
    if (n > m) {
        i = partition(m, n);
        quicksort(m, i-1);
        quicksort(i+1, n);
    }
}
main() {
    readArray();
    a[0] = -9999;
    a[10] = 9999;
    quicksort(1,9);
}
```

Figure 7.2: Sketch of a quicksort program

```
enter main()
    enter readArray()
    leave readArray()
    enter quicksort(1,9)
        enter partition(1,9)
        leave partition(1,9)
        enter quicksort(1,3)
            ...
        leave quicksort(1,3)
        enter quicksort(5,9)
            ...
        leave quicksort(5,9)
    leave quicksort(1,9)
leave main()
```



- Each node represents an activation of a procedure
- The root represents the activation of the main program
- The node for *a* is the parent of the node for *b* if and only if the control flows from activation of *a* to *b*, and
- The node for *a* is to the left of the node for *b* if and only if the lifetime of *a* occurs before the lifetime of *b*.

So, the flow of control in a program corresponds to a depth-first traversal of the activation tree that starts at the root.

- Printing enter when the node for an activation is reached for the first time, and
- Printing leave after the entire subtree of the node has been visited during the traversal.

Overview IV

4.Control stacks

A stack to keep track of live procedure activations:

Push the node for an activation onto the control stack as the activation begins and to pop the node when the activation ends.

Actually, PUSH=ENTER; POP=LEAVE

Overview V

5.Bindings of Names

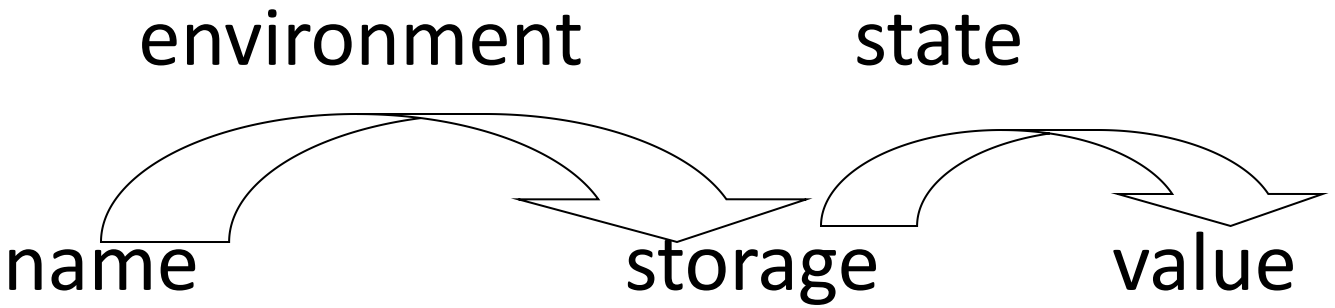
When an environment associates storage location s with a name x, we say that x is bound to s; the association itself is referred to as a binding of x.

Notes:

1)Even each name is declared once in a program, the same name may denote different object at run time

2)The " data object" corresponds to a storage location that can hold values

3)In programming language semantics, the term "environment" refers to a function that maps a name to a storage location, and term "state" refers to a function that maps a storage location to the value held there.



4)Environments and states are different; an assignment changes the state, but not the environment

5)If x is not of a basic type, the storage s for x may be a collection of memory words

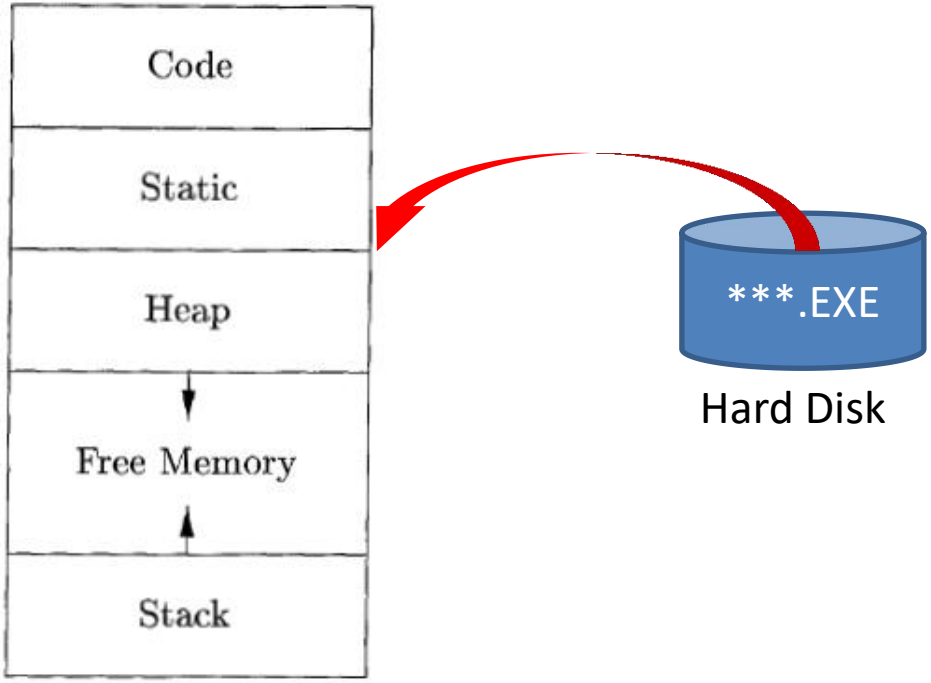
6)A binding is the dynamic counterpart of a declaration

Overview VI

6.Factors that determines the run time environment of a program

- Recursive definition
- Local name storage space allocation strategy
- global name storage space allocation strategy
- Parameter passing mechanism
- A function is whether allowed to be a parameter of or return value of a function

Overview VII



Typical subdivision of run-time memory into code and data areas

Stack Allocation I

- Control Stacks: A stacks of activation records.

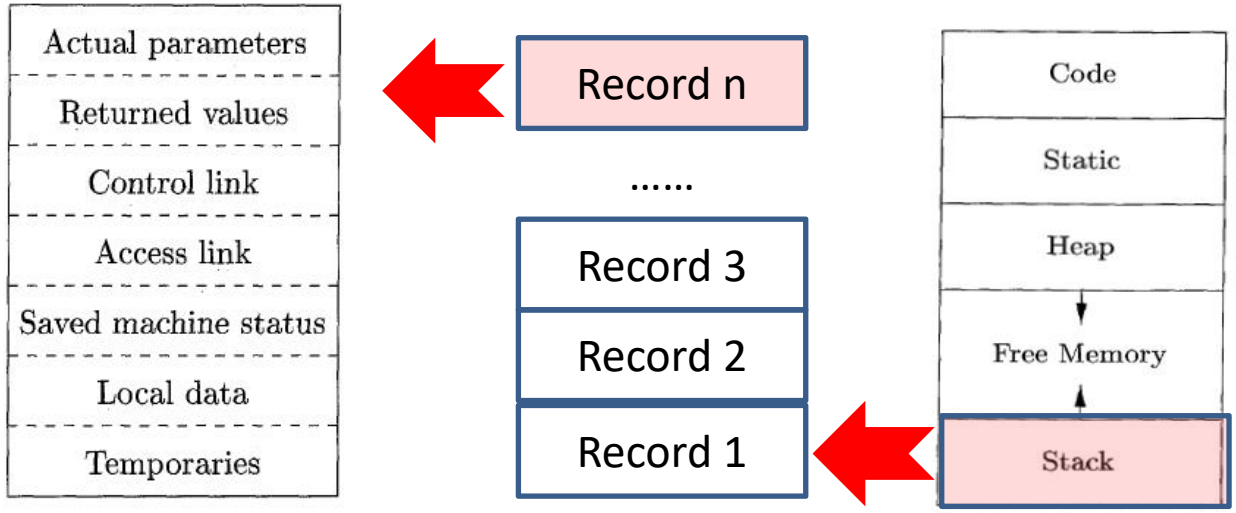


Figure 7.5: A general activation record

Typical subdivision of run-time memory into code and data areas

1. Temporary values, such as those arising from the evaluation of expressions, in cases where those temporaries cannot be held in registers.

2. Local data belonging to the procedure whose activation record this is.

3. A saved machine status, with information about the state of the machine just before the call to the procedure. This information typically includes the *return address* (value of the program counter, to which the called procedure must return) and the contents of registers that were used by the calling procedure and that must be restored when the return occurs.

4. An "access link" may be needed to locate data needed by the called procedure but found elsewhere, e.g., in another activation record. Access links are discussed in Section 7.3.5.

5. A *control link*, pointing to the activation record of the caller.

6. Space for the return value of the called function, if any. Again, not all called procedures return a value, and if one does, we may prefer to place that value in a register for efficiency.

7. The actual parameters used by the calling procedure. Commonly, these values are not placed in the activation record but rather in registers, when possible, for greater efficiency. However, we show a space for them to be completely general.

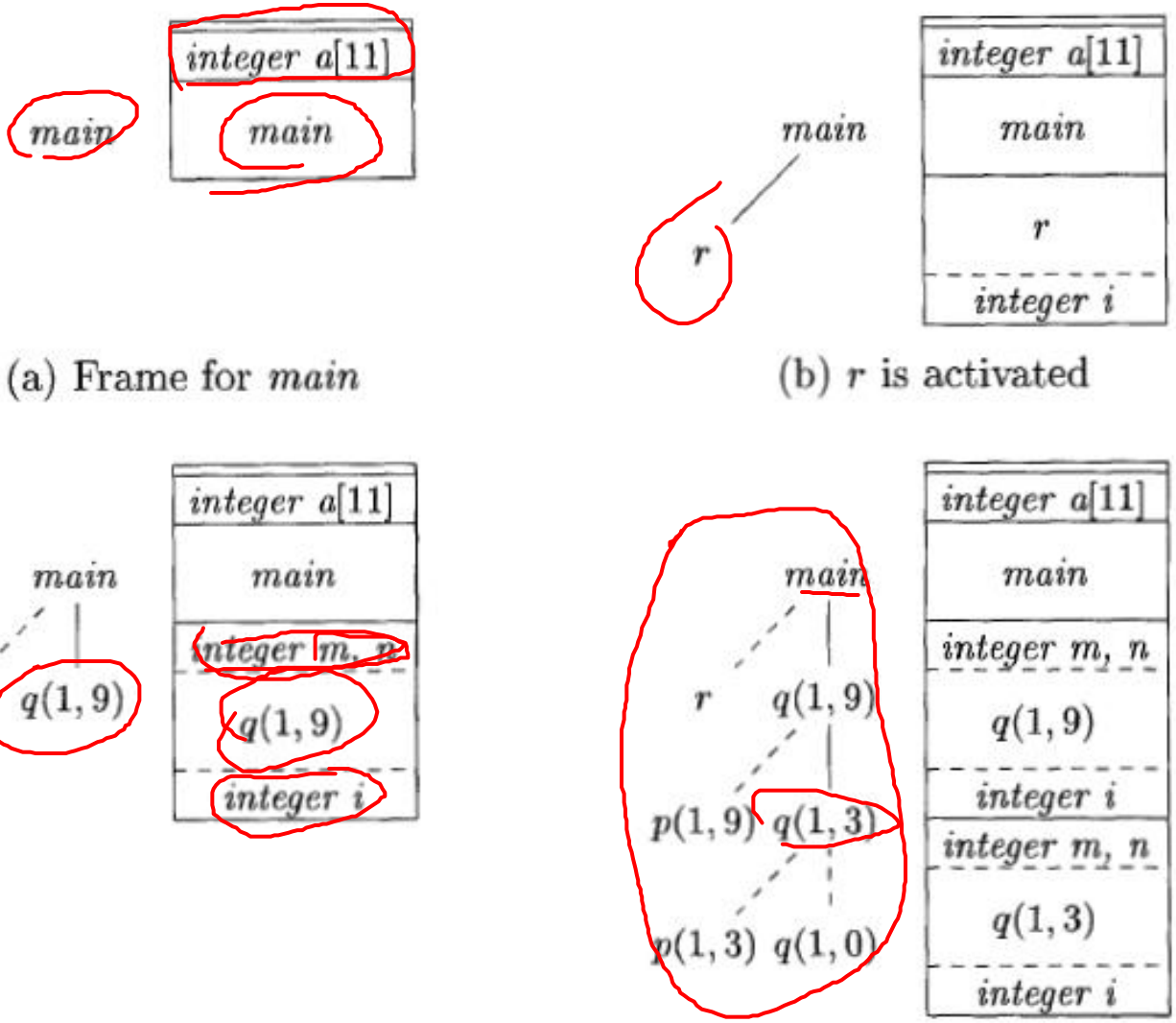


Figure 7.6: Downward-growing stack of activation records

Stack Allocation II

- Calling Sequence.

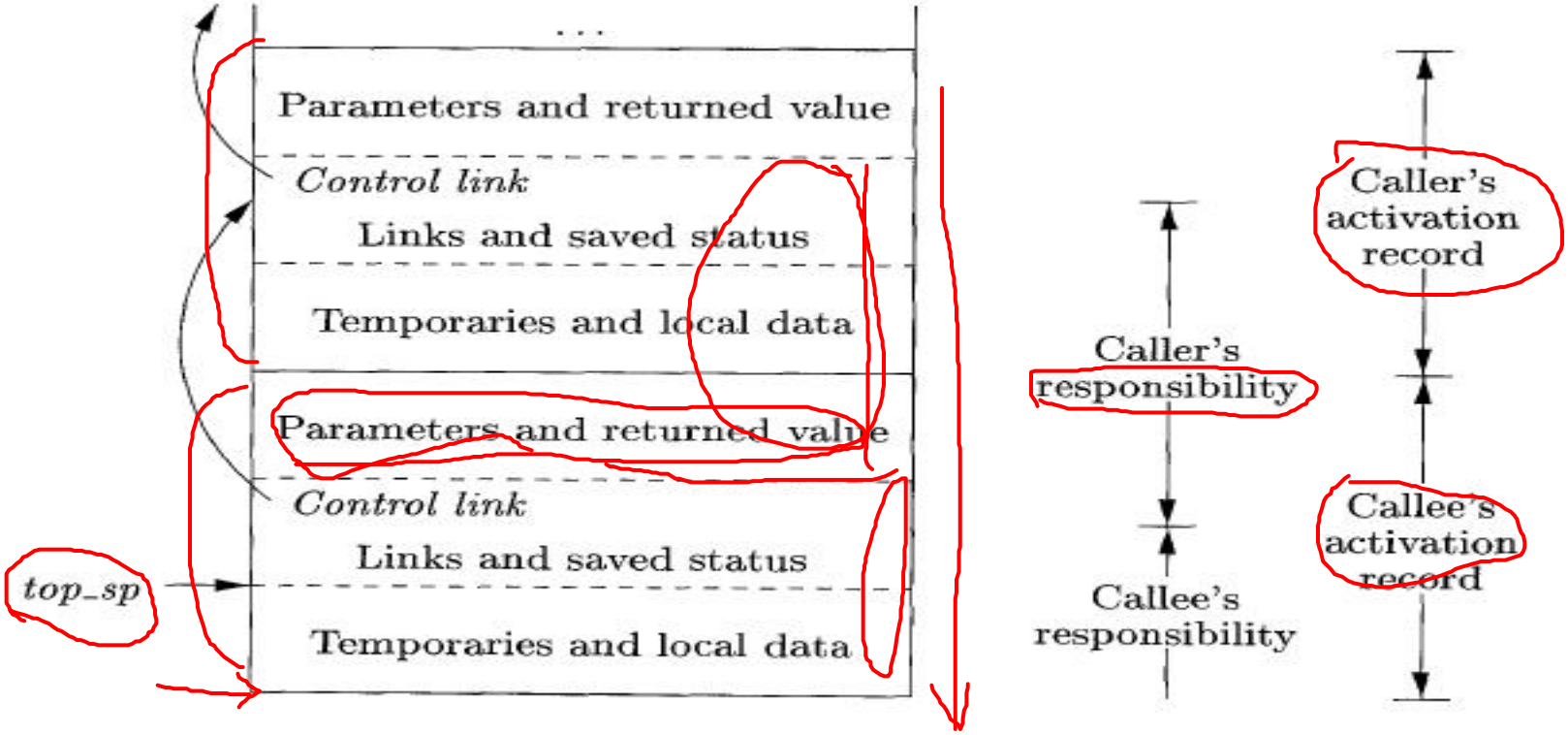


Figure 7.7: Division of tasks between caller and callee

Steps:

(1)The caller evaluates actual parameters.

(2)The caller stores a return address and the old value of top-sp into the callee's activation record.

(3)The callee saves register values and other status information

(4)the callee initializes its local data and begins execution.

A possible return sequence :

- (1)The callee places a return value next to the activation record of the caller
- (2)Using the information in the status field, the callee restores top-sp and other registers and branches to a return address in the caller's code.
- (3)Although top-sp has been decremented, the caller can copy the returned value into its own activation record and use it to evaluate an expression.

Stack Allocation III

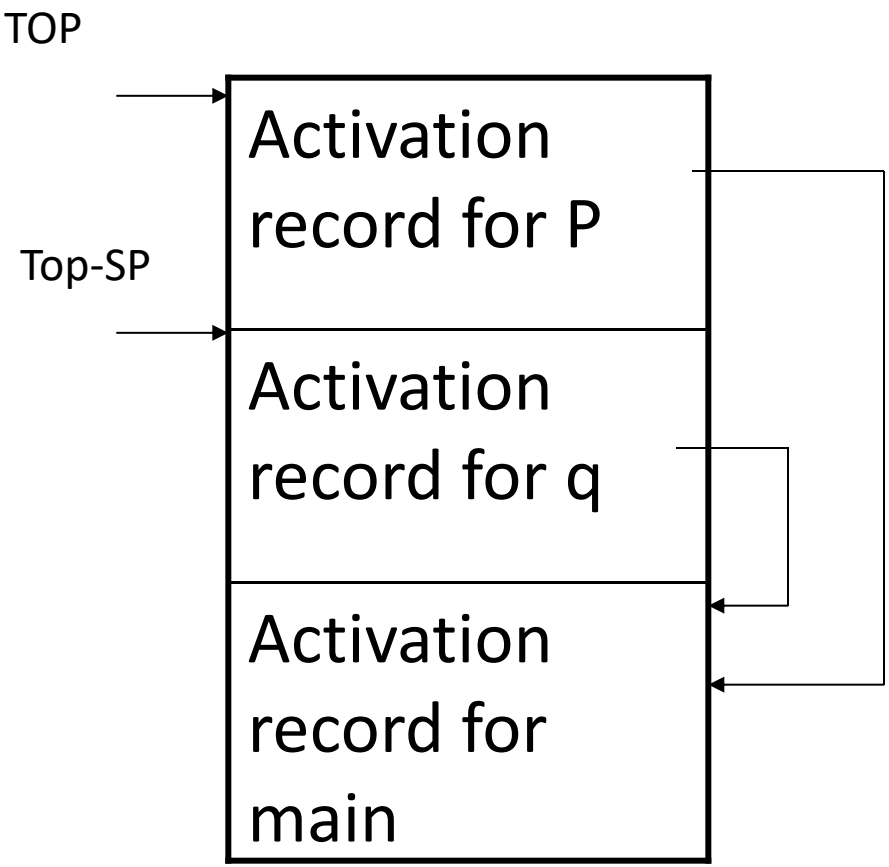
- For non-nested Procedure – C

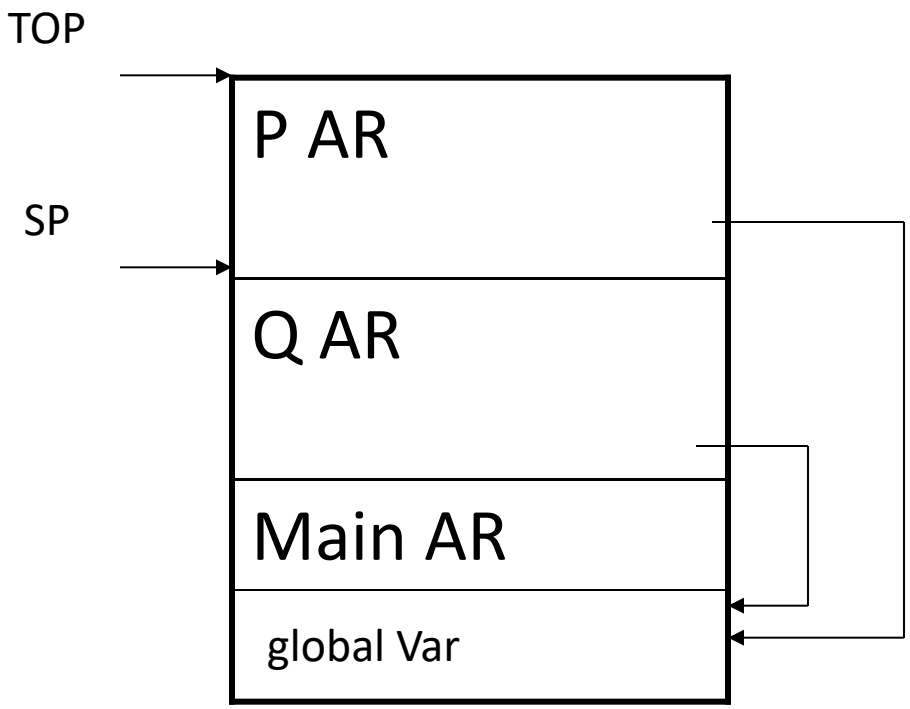
In C, a procedure definition cannot appear within another. If there is a nonlocal reference to a name *a* in some function, then *a* must be declared outside any function.

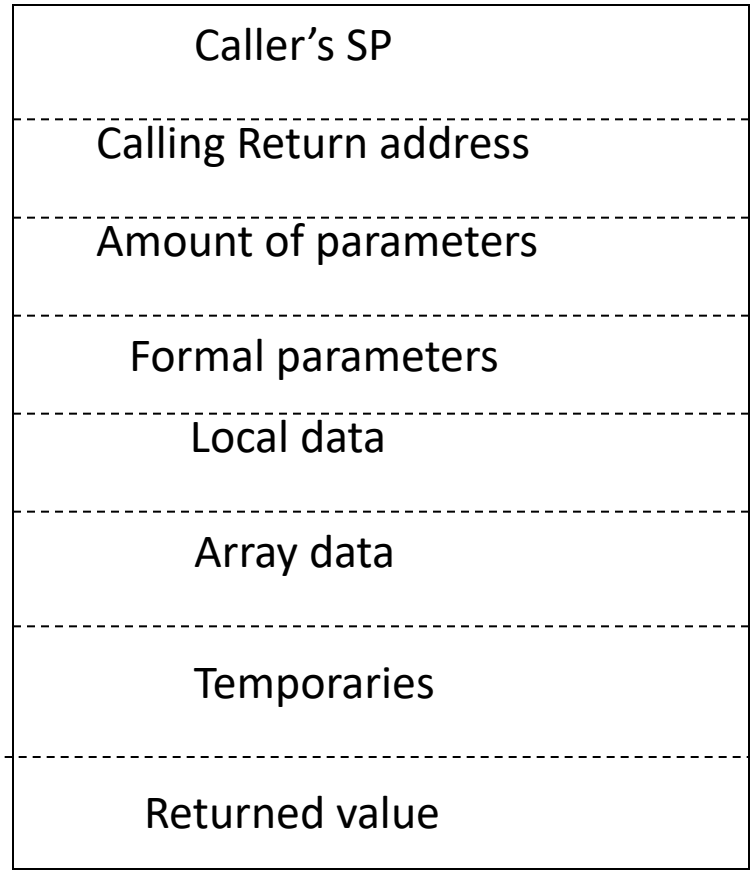
- Stack allocation for C
- When entering a procedure, the activation record of the procedure is set up on the top of the stack
- Initially, put the global(static) variables and the activation record of the Main function

e.g. the calling sequence is like the following:

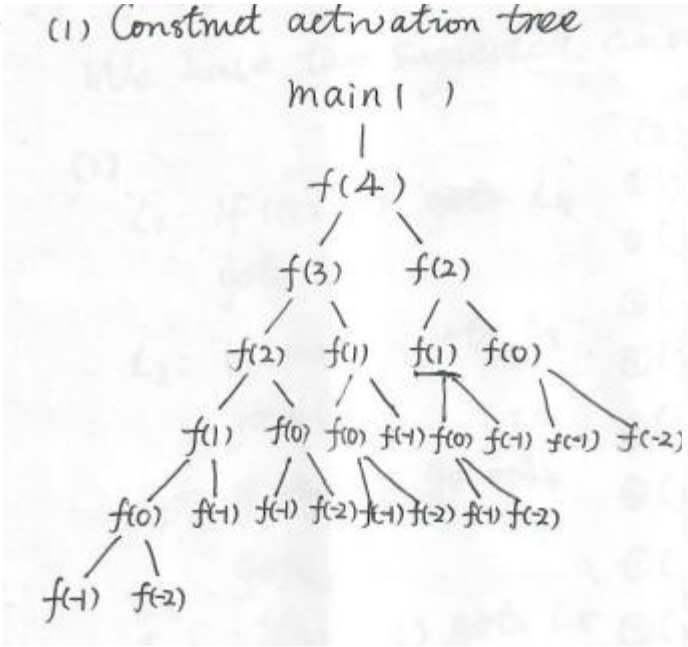
```
main ()
 {……
   q(  );
 }
p(   )
  {……}
q(   )
  {
   ……
   p(   );
   ……
   }
```

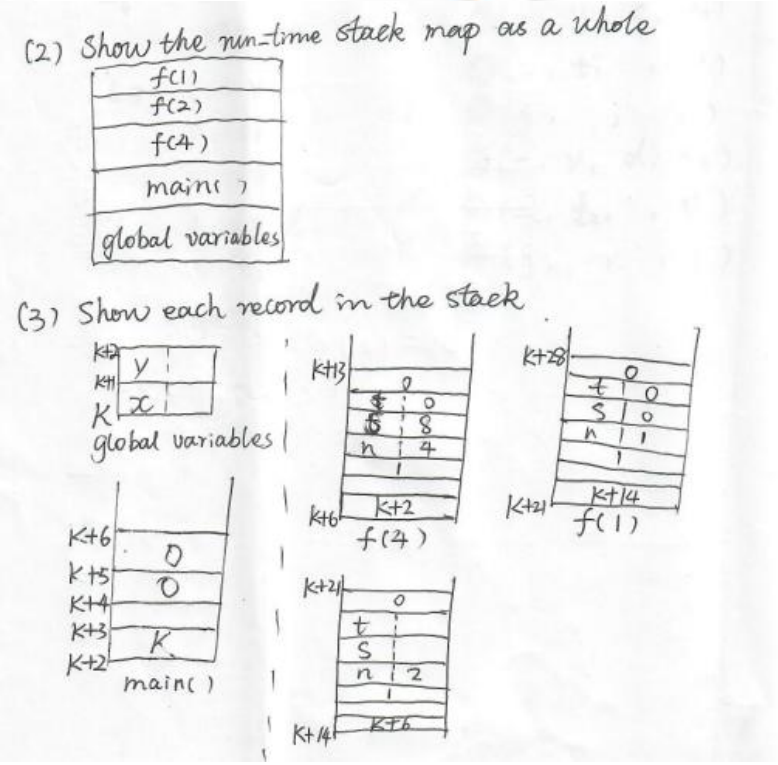

TOP
Top-SP
Activation record for P
Activation record for q
Activation record for main


TOP
SP
P AR
Q AR
Main AR
global Var


Caller’s SP
Calling Return address
Amount of parameters
Formal parameters
Local data
Array data
Temporaries
Returned value

```
#include <stdio.h>
int x,y;
 int main()
 {
    x=4;
    y=f(x);
 }
 int f(int n){
      int t, s;
      if(n<0) return 1;
      s = f(n-1);
      t = f(n-2);
      return s+t;
 }
```



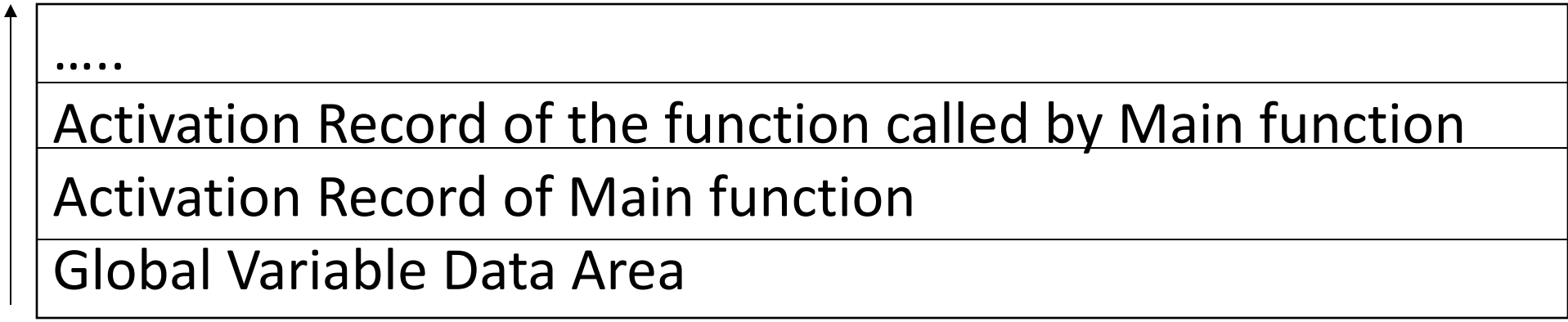


Notes: 1) Here it is assumed that the caller's sp of Main function is the start address of global variable data area, and the returned address in the activation record of a function (including Main function) is filled by the operating system automatically, so you might not care about it.

```
#include <stdio.h>
int x,y;
int main()
{
  x=5;
  y=f(x);
}
```

```
int f(int n)
{
  if (n<=1) return 1;
else  {
   int t1,t2,t;
   t1=f(n-1);
   t2=f(n-2);
t=t1+t2;
   return t} }
```

Storage Organization of C Language



The C program is as the following:

```
#include <stdio.h>
int x,y;
int main()
{ x=5; y=f(x);}
int f(int n)
{ if (n<=1)
      return 1;
else
if ( n==2) return 2;
      else
{ int t1,t2,t3,t4,t;
  t1=f(n-1); t2=f(n-2); t3=f(n-3); t4=t1+t2;t=t3+t4; return t
}
 }
```

Exercise:

construct the run-time stack map when it gets the maximum size for the first time

Notes: 1) Here we assume that the caller's sp of Main function is the start address of global variable data area, and the returned address in the activation record of a function (including Main function) is filled by the operating system automatically, you might not care it.

2) The initial value of variable X is 5, the start address of stack used in the program is K.

3) The stack map may get its maximum size for several times, here we ask you draw the stack map at maximum size for the first time.

Stack Allocation III

- For nested Procedure

Heap Allocation

Garbage Collection

Written Assignment

Please **construct the run-time stack map (detailed map) when it gets the maximum size for the second time** for the following C program

Notes: 1) Here we assume that the caller's sp of Main function is the start address of global variable data area, and the returned address in the activation record of a function (including Main function) is filled by the operating system automatically, you might not care it.

2) The initial value of variable X is 5, the start address of stack used in the program is K.

3) The stack map may get its maximum size for several times, here we ask you draw the stack map (detailed map) at maximum size for the second time.

```
#include <stdio.h>
int x,y;
int main()
{
  x=5;
  y=f(x);
}
```

```
int f(int n)
{
  if (n<=1)
      return 1;
else
if ( n==2)
return 2;
else
  if (n==3)
    return 3;
        else
{
      int t1,t2,t3,t4,t5,t6,t;
      t1=f(n-1);
      t2=f(n-2);
      t3=f(n-3);
      t4=f(n-4)
      t5=t1+t2
t6=t5+t3;
t=t6+t4
      return t
}
 }
```