Unit – 6
Run-Time Environments

Topics to be covered

- Source language issues
- Storage organization
- Storage allocation strategies

Source language issues

Run-Time Environments

- As execution proceeds, the same name is the source text can be denote different data objects in the target machine.
- Each execution of a procedure is referred to as an activation of the procedure.
- If the procedure is recursive, several of its activation may be alive at same time.

Source language issues:

1. Procedures
2. Activation tree
3. Control stack
4. The Scope of a declaration
5. Binding of names

Procedures

- A program is made up of procedures.
- Procedure: declaration that associate an identifier with a statement.
- Identifier: procedure name
- Statement: procedure body
- Procedure call: procedure name appears within an executable statement.
- Example:

```
main()
{
        int n;
        readarray();
        quicksort(1, n);
}
```

```
void quicksort(int m, int n)
{
        int i=partition(m, n);
        quicksort(m, i-1);
        quicksort(i+1, n);
}
```

```
void readarray()
{
        ………
}
int partition(int y, int z)
{
        ……
}
```

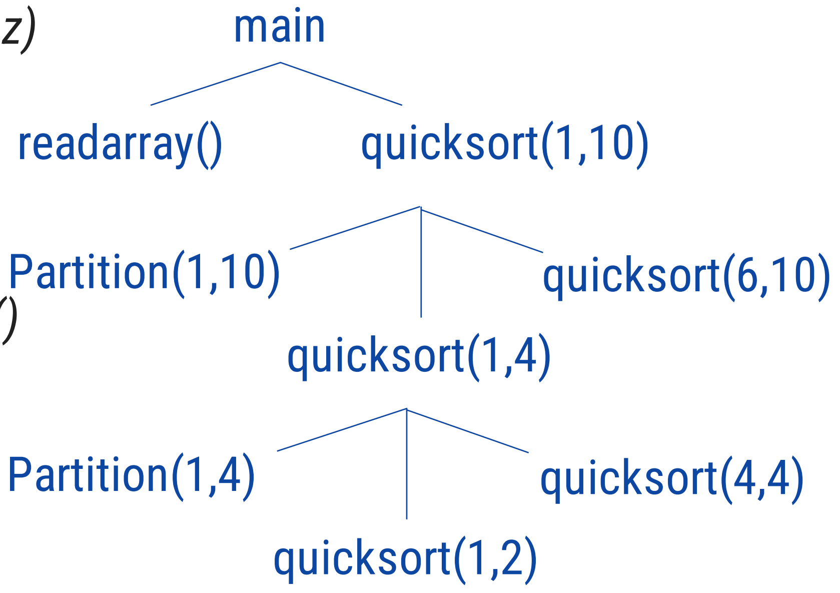
Activation Tree

- If 'a' and 'b' are two procedures, their activations will be:
  - Non-overlapping: when one is called after other
  - Nested: nested procedure
  - Recursive procedure: new activation begins before an earlier activation of the same procedure has ended.
- An activation tree shows the way control enters and leaves activations.
- Properties of activation trees are :-

1. Each node represents an activation of a procedure.
2. The root shows the activation of the main function.
3. The node for procedure 'a' is the parent of node for procedure 'b' if and only if the control flows from procedure a to procedure b.
4. If node 'a' is left of the node 'b' if and only if the lifetime of a occurs before the lifetime of b.

```
main()
{
int n;
readarray();
quicksort(1, n);
}

void quicksort(int m, int n)
{
int i=partition(m, n);
quicksort(m, i-1);
quicksort(i+1, n);
}

void readarray()
{
………
}
int partition(int y, int z)
{
……
}
```

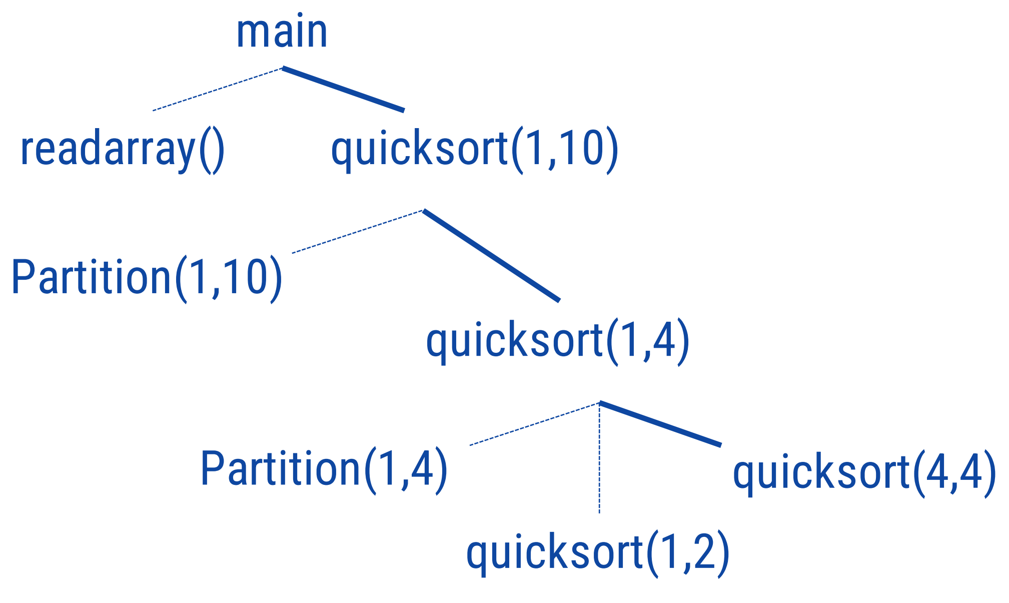
```
main()
{
int n;
readarray();
quicksort(1, n);
}

void quicksort(int m, int n)
{
int i=partition(m, n);
quicksort(m, i-1);
quicksort(i+1, n);
}

void readarray()
{
………
}
int partition(int y, int z)
{
……
}
```



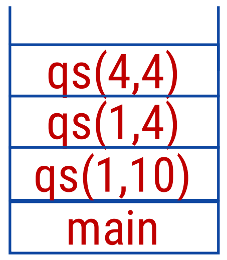
Activation Tree

Control stack

- Control stack or runtime stack is used to keep track of the live procedure activations i.e the procedures whose execution have not been completed.
- A procedure name is pushed on to the stack when it is called (activation begins) and it is popped when it returns (activation ends).
- Information needed by a single execution of a procedure is managed using an activation record or frame. When a procedure is called, an activation record is pushed into the stack and as soon as the control returns to the caller function the activation record is popped.



Activation Tree



Scope of a declaration

- A declaration in a language is a syntactic construct that associate information with a name.
  - Var i: integer;
- There may be declaration of the same name in different parts of a program.
- The scope rules of a language determine which declaration of a name applies when the name appears in the text of a program.
- The portion of the program, to which declaration applies is called the scope of the declaration.
- An occurrence of a name in a procedure is said to be local to the procedure if it is in the scope of a declaration within the procedure; otherwise the occurrence is said to be nonlocal.
- The distinction between local and non local names carries over to any syntactic construct that can have declaration within it.

Binding of names

- Environment: function that maps a name to a storage location.
- State: function that maps a storage location to the value held there.

$$\begin{tikzcd}
\text{Name} \arrow[r, bend left, "\text{Environment}"] & \begin{array}{c}\text{Storage}\\ \text{(l-value)}\end{array} \arrow[r, bend left, "\text{State}"] & \begin{array}{c}\text{Value}\\ \text{(r-value)}\end{array}
\end{tikzcd}$$

- When an environment associates storage location s with a name x→x is bound to s.

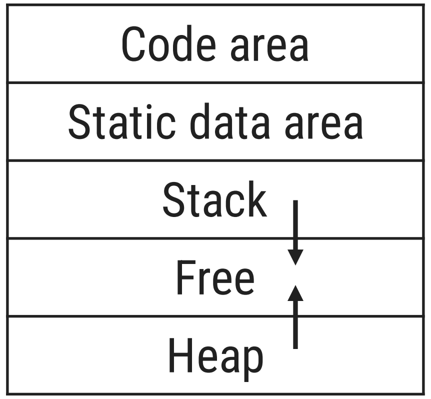
Storage Organization

Storage organization

- The executing target program runs in it's own logical address space in which each program value has a location.
- The management and organization of this logical address space is shared between the compiler, operating system and target machine.
- The operating system maps the logical address into the physical address, which are usually spread throughout memory.

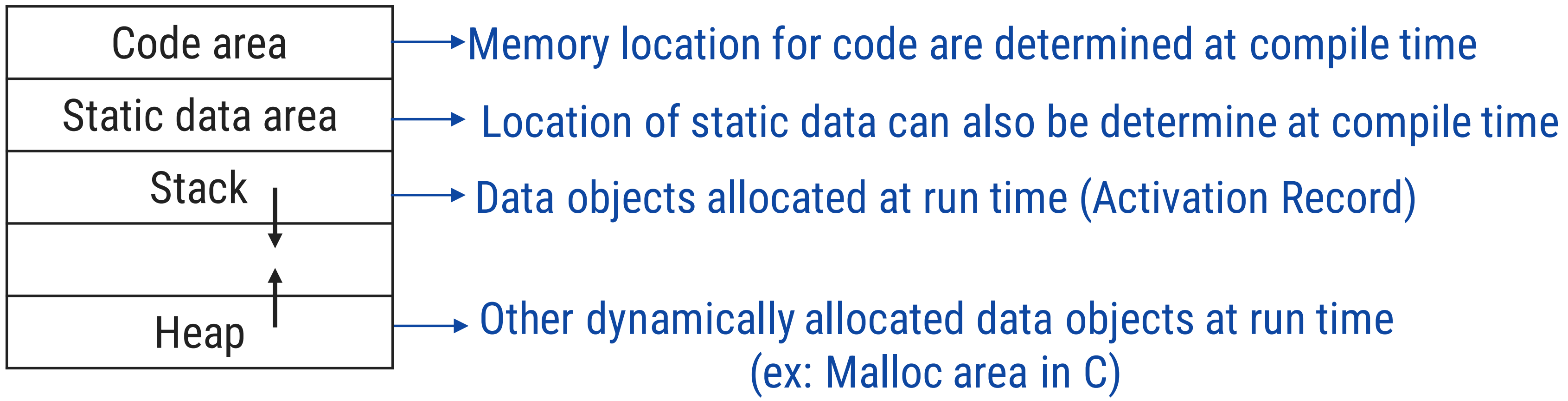
Subdivision of Runtime Memory

- The compiler demands for a block of memory to operating system.
- The compiler utilizes this block of memory executing the compiled program. This block of memory is called **run time storage**.
- The run time storage is subdivided to hold code and data such as, the generated target code and data objects.
- The size of generated code is fixed. Hence the target code occupies the determined area of the memory.



Subdivision of Runtime Memory

- Code block consisting of a memory location for code.
- The amount of memory required by the data objects is known at the compiled time and hence data objects also can be placed at the statically determined area of the memory.
- Stack is used to manage the active procedure.
- Managing of active procedures means when a call occurs then execution of activation is interrupted and information about status of the stack is saved on the stack.
- Heap area is the area of run time storage in which the other information is stored.


Code area
Static data area
Stack
Heap
Memory location for code are determined at compile time
Location of static data can also be determine at compile time
Data objects allocated at run time (Activation Record)
Other dynamically allocated data objects at run time
(ex: Malloc area in C)

Activation Record

- Control stack is a run time stack which is used to keep track of the live procedure activations i.e. it is used to find out the procedures whose execution have not been completed.
- When it is called (activation begins) then the procedure name will push on to the stack and when it returns (activation ends) then it will popped.
- Activation record is used to manage the information needed by a single execution of a procedure.
- An activation record is pushed into the stack when a procedure is called and it is popped when the control returns to the caller function.

Activation Record

- The execution of a procedure is called its activation.
- An activation record contains all the necessary information required to call a procedure.

- **Return value:** used by the called procedure to return a value to calling procedure.
- **Actual parameters:** This field holds the information about the actual parameters.
- **Control link (optional):** points to activation record of caller.
- **Access link (optional):** refers to non-local data held in other activation records.
- **Machine status:** holds the information about status of machine just before the function call.
- **Local variables:** hold the data that is local to the execution of the procedure.
- **Temporary values**: stores the values that arise in the evaluation of an expression.

Return value
Actual parameters
Control link
Access link
Machine status
Local variable
Temporary values

Compile-Time Layout of Local Data

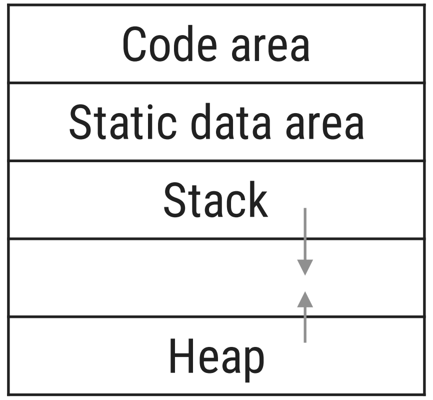
- The amount of storage needed for a name is determined from its type. (e.g.: int, char, float…)
- Storage for an aggregate, such as an array or record, must be large enough to hold all it’s components.
- The field of local data is laid out as the declarations in a procedure are examined at compile time.
- We keep a count of the memory locations that have been allocated for previous declarations.
- From the count we determine a relative address of the storage for a local with respect to some position such as the beginning of the activation record.

Storage allocation strategies

Storage allocation strategies

The different storage allocation strategies are:

- Static allocation: lays out storage for all data objects at compile time.
- Stack allocation: manages the run-time storage as a stack.
- Heap allocation: allocates and de-allocates storage as needed at run time from a data area known as heap.



Static allocation

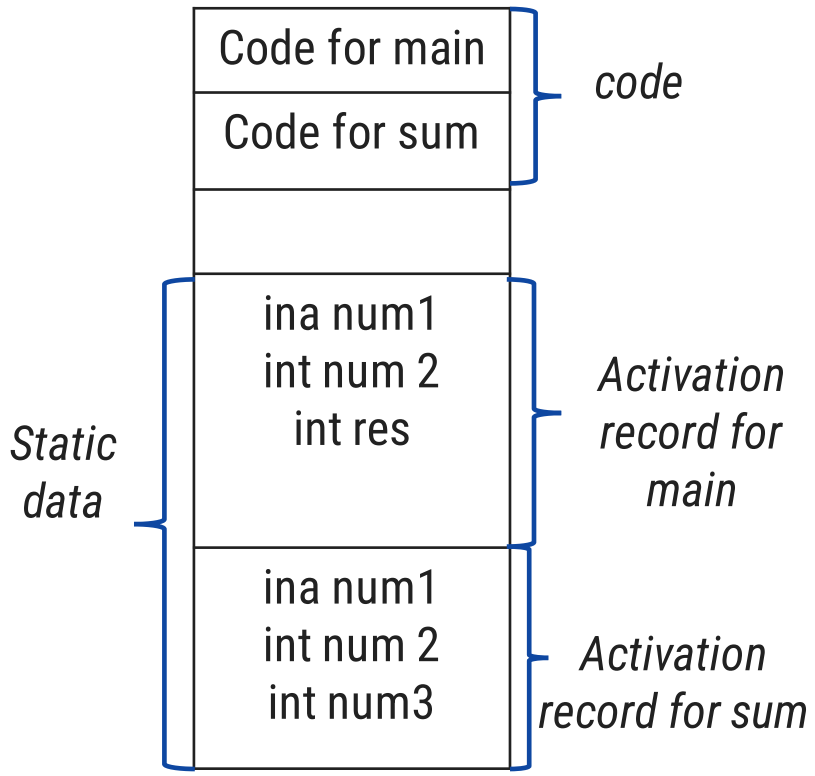
- In static allocation, names are bound to storage as the program is compiled, so there is no need for a run-time support package.
- Since the bindings do not change at run-time, every time a procedure is activated, its names are bounded to the same storage location.

Limitation

1. Size of data object must be known at compile time.
2. Recursive procedures are restricted.
3. Data structure can't be created dynamically.

Example:

```
int main()
{
  int num1=100, num2=200, res;
  res = sum(num1, num2);
  printf("\n Addition is %d : ",res);
  return (0);
}
 int sum(int num1, int num2)
{
  int num3;
  num3 = num1 + num2;
  return (num3);
}
```

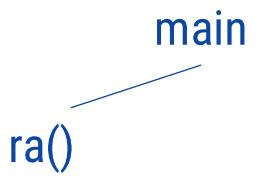
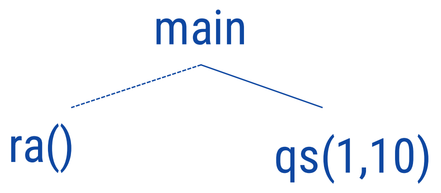


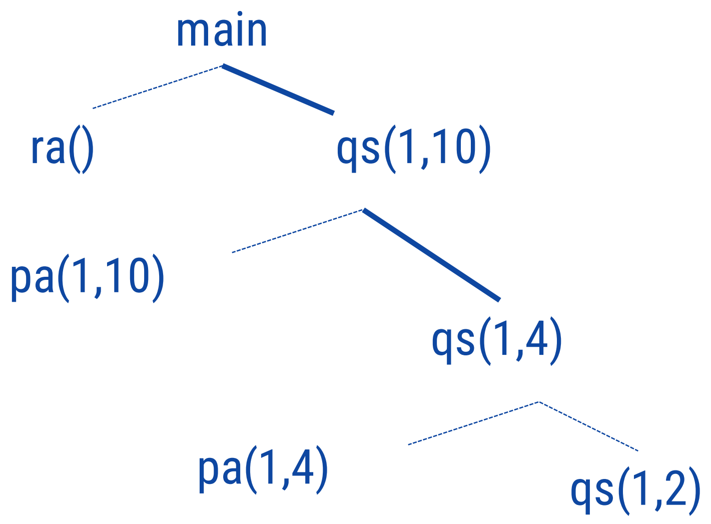
Stack allocation

- All compilers for languages that use procedures, functions or methods as units of user define actions manage at least part of their run-time memory as a stack.
- Each time a procedure is called, space for its local variables is pushed onto a stack, and when the procedure terminates, the space is popped off the stack.
- Locals are bound to fresh storage in each activations.
- Locals are deleted when the activation ends.

Stack allocation

- At run time, an activation record can be allocated by incrementing ‘top’ by the size of the record.
- Deallocated by decrementing ‘top’ by the size of record.

Position in activation tree	Activation record on the stack	Remarks
main	main Int n	Frame for main
	main Int n ra Int i	Ra is activated
	main Int n qs(1,10) Int i	Frame ra has been poped and qs(1,10) is pushed.

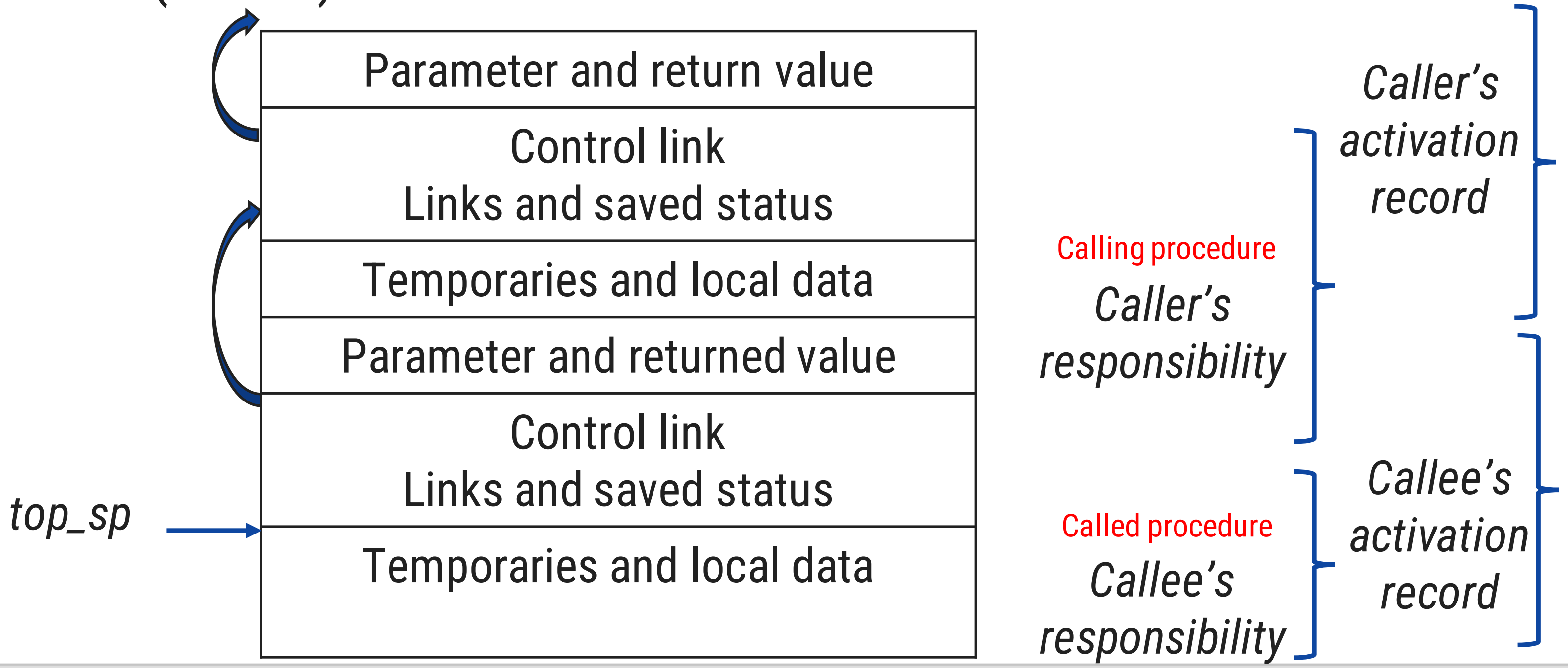


Activation Tree

main
Int n
qs(1,10)
Int i
qs(1,4)
Int i

Stack allocation: Calling Sequences

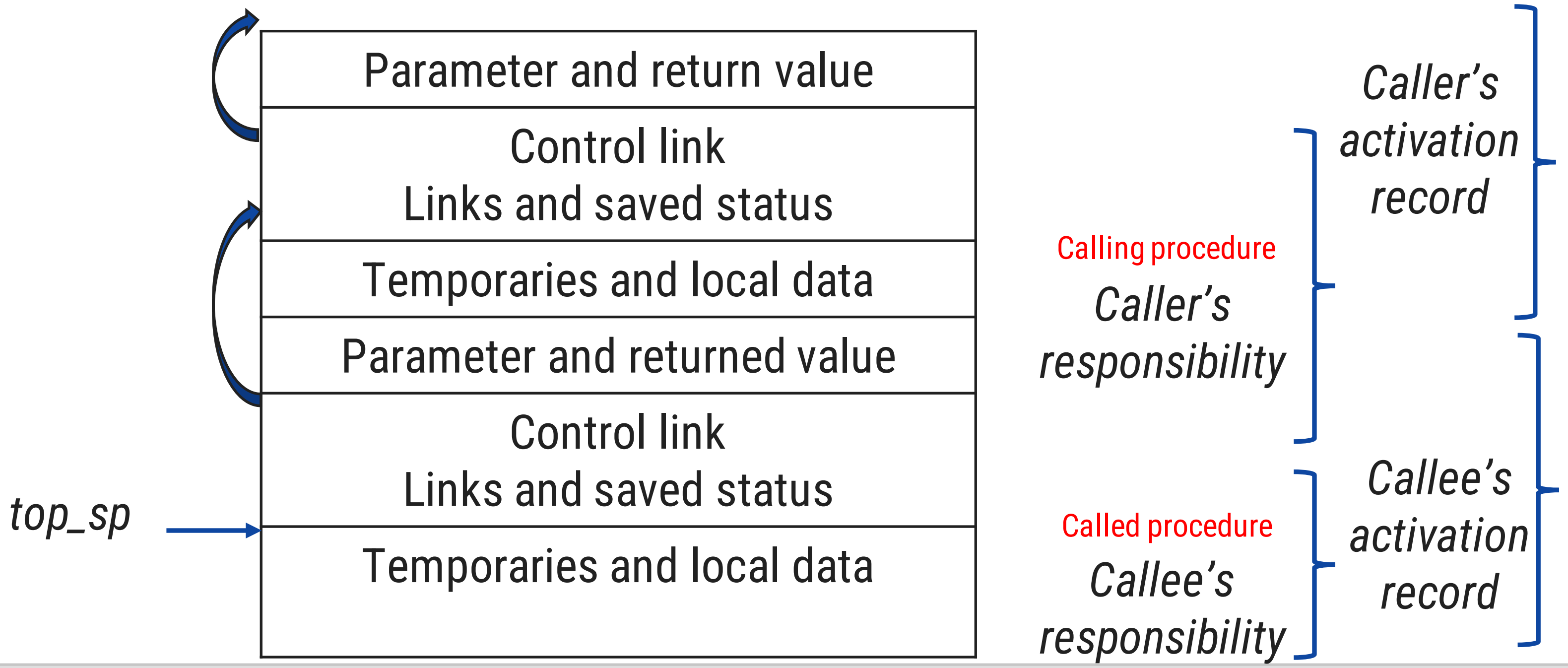
- Procedures calls are implemented by generating what are known as calling sequences in the target code.
- A call sequence allocates an activation record and enters the information into its fields.
- A Return sequence restore the state of machine so the calling procedure can continue its execution.
- The code is calling sequence of often divided between the calling procedure (caller) and procedure is calls (callee).



Parameter and return value
Control link
Links and saved status
Temporaries and local data
Parameter and returned value
Control link
Links and saved status
Temporaries and local data
top_sp
Calling procedure
Caller's responsibility
Called procedure
Callee's responsibility
Caller's activation record
Callee's activation record

Stack allocation: Calling Sequences

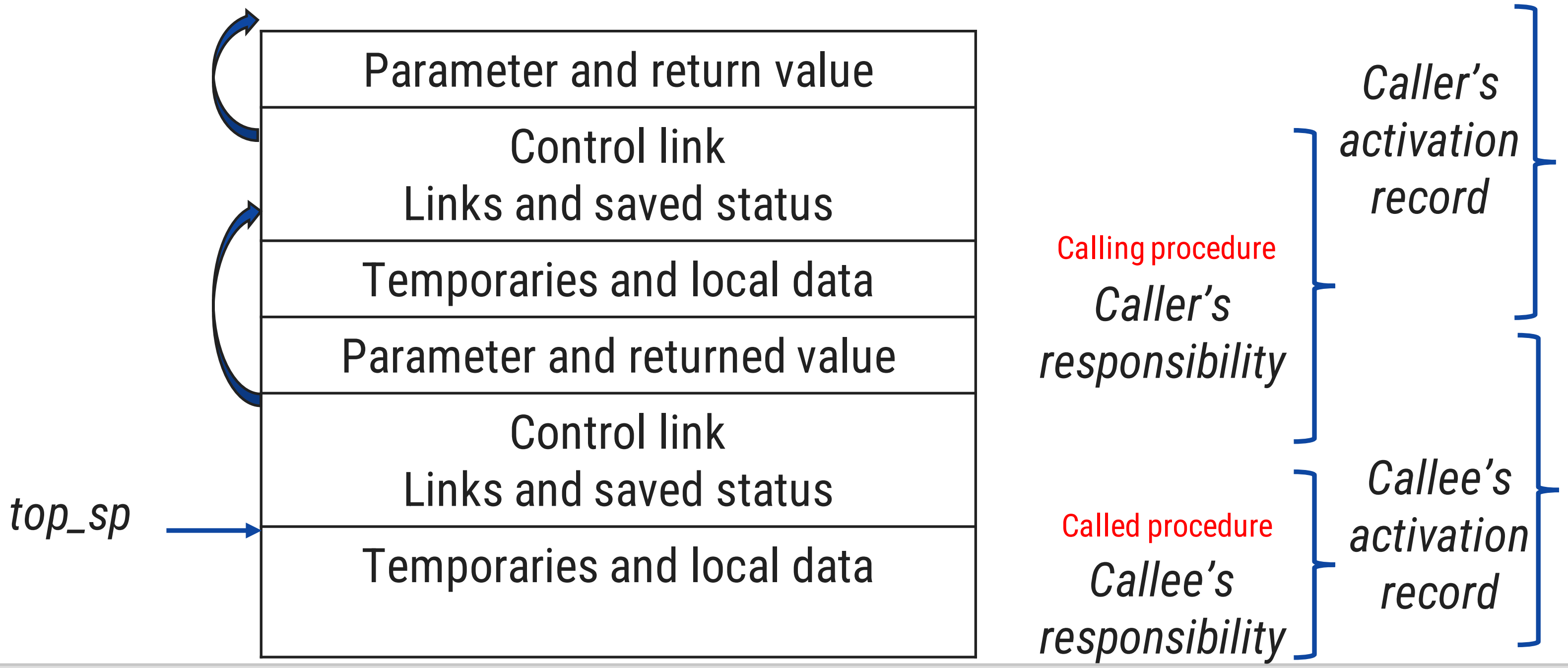
- The calling sequence and its division between caller and callee are as follows:
  1. The caller (Calling procedure) evaluates the actual parameters.
  2. The caller stores a return address and the old value of *top_sp* into the callee's activation record. The caller then increments the *top_sp* to the respective positions.
  3. The callee (Called procedure) saves the register values and other status information.
  4. The callee initializes its local data and begins execution.



Parameter and return value
Control link
Links and saved status
Temporaries and local data
Parameter and returned value
Control link
Links and saved status
Temporaries and local data
top_sp
Caller's activation record
Calling procedure
Caller's responsibility
Callee's activation record
Called procedure
Callee's responsibility

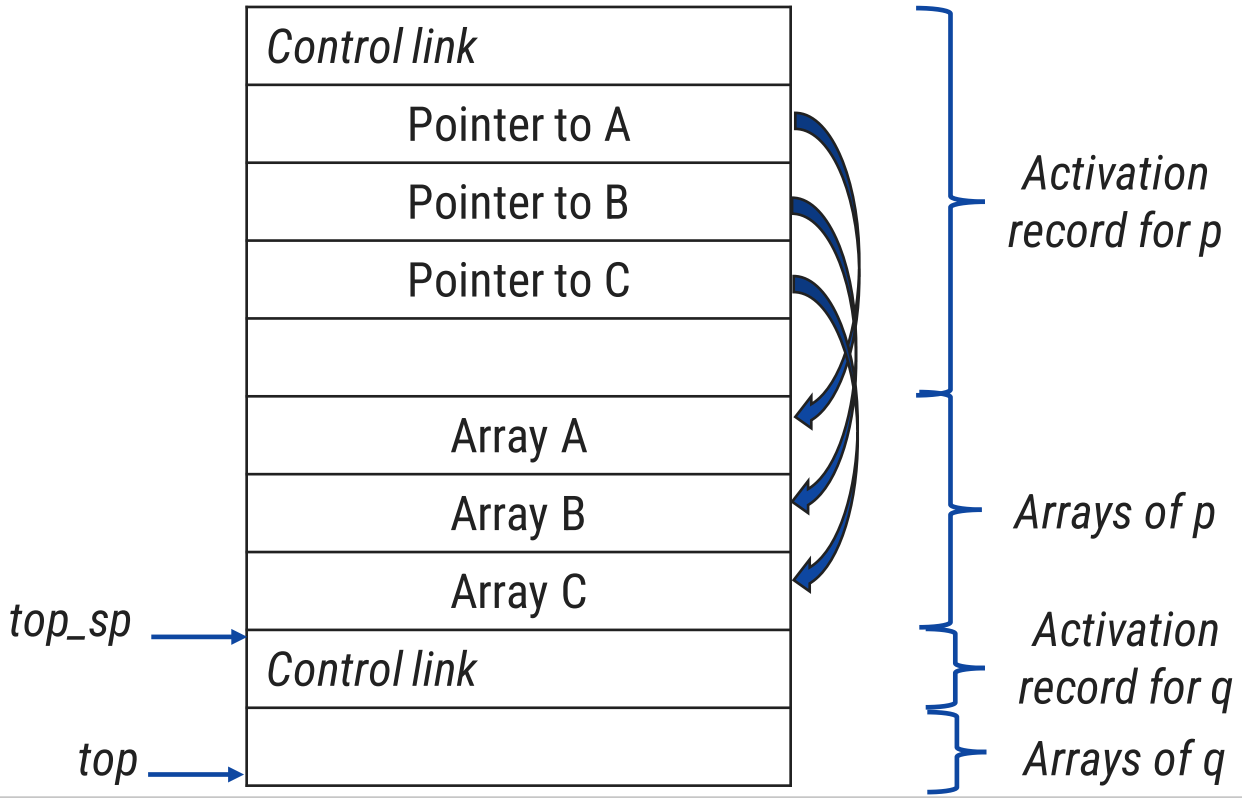
Stack allocation: Calling Sequences

- A suitable, corresponding return sequence is:
  1. The callee places the return value next to the parameters.
  2. Using the information in the machine status field, the callee restores *top_sp* and other registers, and then branches to the return address that the caller placed in the status field.
  3. Although *top_sp* has been decremented, the caller knows where the return value is, relative to the current value of *top_sp* ; the caller therefore may use that value.


Parameter and return value
Control link
Links and saved status
Temporaries and local data
Parameter and returned value
Control link
Links and saved status
Temporaries and local data
top_sp
Caller's activation record
Calling procedure
Caller's responsibility
Callee's activation record
Called procedure
Callee's responsibility

Stack allocation: Variable length data on stack

- The run time memory management system must deal frequently with the allocation of objects, the sizes of which are not known at the compile time, but which are local to a procedure and thus may be allocated on the stack.
- The same scheme works for objects of any type if they are local to the procedure called have a size that depends on the parameter of the call.



top: position of the next activation record

top_sp: find the local data

Stack allocation: Dangling Reference

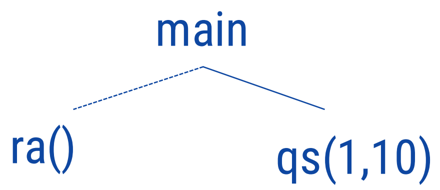
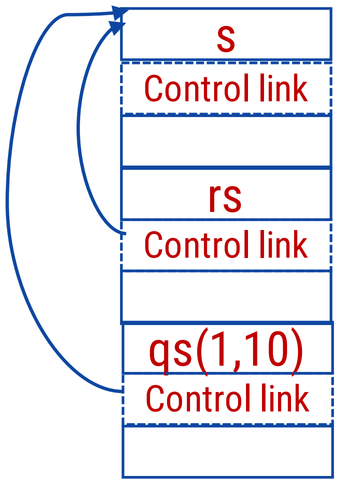
- When there is a reference to storage that has been deallocated.
- It is a logical error to use dangling reference, since, the value of de-allocated storage is undefined according to the semantics of most languages.

```
main()
{
        int *p;
        p=dengle();
}
int *dengle()
{
        int i=10;
        return &i;
}
```

Heap Allocation

The stack allocation strategy can not be used for following condition:

1. The values of the local names must be retained when an activation ends.
2. A called activation outlives the called.

Position in activation tree	Activation record on the heap	Remarks
	 s Control link rs Control link qs(1,10) Control link	Retains activation record for ra

- Records for the live activations need not be adjacent in a heap.

References

Books:

1. **Compilers Principles, Techniques and Tools, PEARSON Education (Second Edition)**

   Authors: Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman

2. **Compiler Design, PEARSON (for Gujarat Technological University)**

   Authors: Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman

Thank You