8ta

E2·

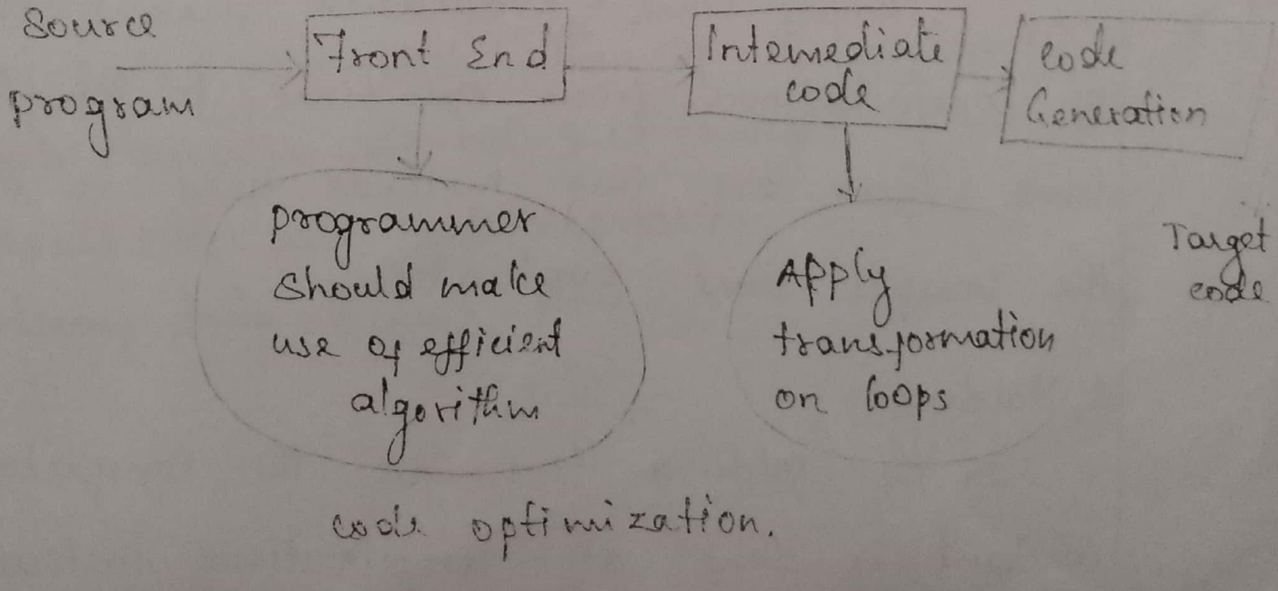
Unit : 5
Code Generation.

Introduction to Optimization Techniques - The principle sources of optimization - Issues in the design of code generator - Run time storage Management - static allocation - Stack allocation - Design of a simple code generator.

Introduction to Optimization Techniques:

The code optimization is required to produce an efficient target code. There are two important issues that need to be considered while applying the techniques for code optimizati They are.

* Semantic equivalence should be Maintain
* Algorithm should not be changed.



Principle Sources of Optimization:.

The optimization can be done locally, globally. If the transformation is applied on the same basic block then that kind of transformation is done locally otherwise it is done globally.

Function Preserving transformation

There are a number of ways in which a compiler can improve a program without changing the function it computes.

eg: * Common Subexpression elimination
* Copy propagation
* Dead-code elimination
* Constant folding.

* Compile Time Evaluation:.

Compile time evaluation means shifting of computations from run time to compilation time. There are two methods used to obtain the compile time evaluation.

1. Folding:.

In the folding technique the computation of constant is done at compile time instead of execution time.

eg: $l = (22/7) * d$.

Here folding is implied by performing the computation of 22/7 at compile time instead of execution time.

2. Constant Propagation:

In this technique the value of variable is replaced and computation of an expression is done at the compilation time.

eg: pi = 3.14
r = 5
Area = pi * r * r

The value of pi & r is replaced and executed & at the compilation time.

Common Subexpression Elimination:

The common subexpression is an expression appearing repeatedly in the program which is computed previously. Then if the oprands of this sub expression do not get changed at all then result of such sub expression is used instead of recomputing it each time.

eg: $t_1 := 4 * i$

$t_2 := a[t_1]$

$t_3 := 4 * j$

$t_4 := 4 * i$

$t_5 := n;$

$t_6 := b[t_4] + t_5$

The above code can be optimized using the common sub expression elimination as,

$t_1 := 4 * i$

$t_2 := a[t_1]$

$t_3 := 4 * j$

$t_5 := n$

$t_6 := b[t_1] + t_5$.

The common sub expression $t_4 := 4 * i$ is eliminated as its computation is already in t_1 and value of i is not been changed from definition to use.

* Copy Propagation :.

Variable propagation means use of one variable instead of another.

for eg: $x = pi;$

$area = x * r * r;$

The optimization using variable propagation can be done as follows

$area = pi * r * r$

Here the variable x is eliminated.

*Code Movement:·

There are two basic goals of code movemen

*To reduce the size of the code is to obtain the Space complexity.

* To reduce the frequency of execution of code is to obtain the time complexity.

eg:
```
for(i=0; i<=10; i++)
{
    x = y*5;
    ...
    k= (y*5)+50;
}
```

This can be optimized as
```
    z = y*5;
for(i=0; i<=10; i++)
{
    x=z;
    ...
    k=z+50;
}
```

The code for y*5 will be generated only onc And simply the result of that computation is used wherever necessary.

*Strength Reduction:·

The strength of certain operators is higher than others. eg strength of * is higher than +

Scanned by CamScanner

In strength reduction technique the hig
strength operators can be replaced by lo
strength operators.

```
eg: for (i=1; i<=50; i++)
      {
        ....
        count = i x 7;
        ...
      }
```

Here we get the count values as 7, 14, 21 and so on upto less than 50. This code can be replaced by using strength reduction as follows.

```
temp = 7
for (i=1; i<=50; i++)
{
  ....
  count = temp;
  temp = temp + 7;
}
```

* Dead Code Elimination:

A variable is said to be live in a progra
if the value contained into it is used subsequent
~~else~~. the variable is said to be dead at a poin
in program if the value contained into it is never been used. The code containing such a variable supposed to be a dead code.

```
eg:  i=j;
     ...
     ...
     x=i+10;
     ...
```

The optimization can be performed by eliminati the assignment statement i=j. This assignment statement is called dead assignment.

* Loop Optimization :.

The code optimization can be significantly done in loops of the program. Specially inner loop is a place where program spends large amount of time. Hence if no. of instructions are less in inner loop then the running time of the program decreased to a larger extent.

Methods :.

1. Code Motion
2. Induction variable and strength reduction
3. Loop invariant Method
4. Loop unrolling
5. Loop fusion.

* Code Motion :.

→ moves the code outside the loop.

```
eg: while(i<=max-1)            n=max-1;
    {                          while(i<=n)
      sum=sum+a[i];     ⇒      {
    }                          sum=sum+a[i];
                               }
```

Scanned by CamScanner

Induction variables and reduction in St

*Loop

* A variable x is called an induction variable of Loop L if the value of variable changed every time.

* Reducing the strength which is higher than others.

* Loop Invarient Method:

→ Computation inside the loop is avoided and thereby the computation overhead on compiler is avoided.

eg:

```
for i:=0 to 10 do begin              t:= a/b;
    k:=i+a/b;                        for i:=0 to 10 do begin
    ...                     =>           k:=i+t;
    ...                                  ...
    end                                  ...
                                         end
```

* Loop Unrolling:

→ no. of jumps and tests can be reduced by writing the code two ~~ty~~ times.

eg:

```
int i=1;                        int i=1;
while (i<=100)                  while (i<=100)
{                      =>       { a[i]=b[i];
  a[i] = b[i];                    i++;
  i++;                            a[i]=b[i];
}                                 i++;
                                }
```

*Loop fusion or Loop Jamming:

→ Several loops are merged to one loop

eg:

```
for i:=1 to n do              for i:=1 to n*m do
for j:=1 to m do      ⇒       a[i]:=10.
  a[i,j]:=10
```

Issues in Code Generation:

The following are some of the issues in code generation.

1. Input to code Generator.
2. Target Programs
3. Memory Management
4. Instruction Selection – uniformity & completeness of Instruction set depends upon the select of instruction to ~~[illegible]~~ speed up the process.
5. Register Allocation
6. Approaches to code generation
7. Choice of evaluation Order.

① *Input to the code generator:

*The intermediate representation of the source program produced by the front end.

* Several choices for the intermediate languages.

1. Linear – postfix Notation
2. 3 address – Quadruples

3. Viructal Machine :- Stack Machine

4. Graphical - Syntax tree and

* Evaluat

* Memory Management:

③ * Mapping names in the source progra to addresses of data objects in run-time mem

* Done by the front end and the code generator.

* A name in a three-address statement refe to a symbol-table entry for the name

* A relative address can be determined.

② *Target Programs:

1. ~~Absoulte~~ Absolute Machine language
2. Relocateable Machine language
3. Assembly language

③ *Register Allocation:

* Instructions with register operands are usually shorter and faster.

* Efficient utilization of registers is important in generating good code.

1. Register Allocation phase
2. Register Assignment phase.

* Evaluation Order:
7
The order in which computations are performed. Affect the efficiency of the target code. Some Computation orders require fewer registers to hold intermediate results.

6 * Approaches to code generation:

*Most important criteria for code generator is that it produces correct code

* Correctness takes on special significance

* It contains a straightforward code generation algorithm.

* Design of a simple code generator:

→ The computed results can be kept in registers as long as possible.

→ The code generation algorithm uses descriptors to keep track of register contents and address for names.

A register descriptor is used to keep track of what is currently in register.

Attributes	Addressing modes	Storage Location

~~and~~ Address descriptor.

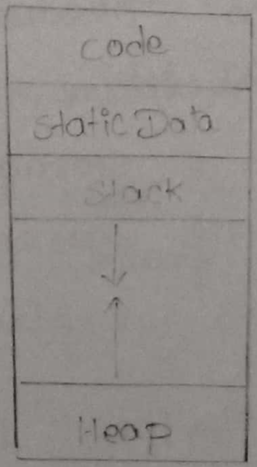
Run-Time
TORAGE ORGANIZATION

The Organization of run-time Storage can be used for languages Such as FORTRAN Pascal and c.

Sub division of Run-Time memory

The Runtime Storage might be a sub divided to hold

1. The generated target code
2. data objects and
3. A counter part of the control stack to keep track of Procedure activations



The size of the generated target code is fixed at compile time, so the compiler can place it in a Statically determined area

Similarly, the size of some of the data o
may also be known at compile time and th
too can be placed in a statically determine
area. One reason for statically allocating as
many data objects as possible is that the
addresses of these objects can be compiled
in to the target code.

the control stack is used to manage.
activations of procedures.

A separate area of run-time memory
called a heap.

ACTIVATION RECORDS

Information needed by a single
execution of a procedure is managed using
a contiguous block of storage called an
activation record or frame, consisting of the
collection of fileds.

The purpose of the fields of an activation record is as follows,

returned value
actual parameters
optional control link
optional access link
saved machine status
local data
temporaries

1. Temporary values are stored in the field for temporaries.
2. The field for local data holds data that is local to an execution of a procedure.
3. The field for saved machine status holds information about the state of the machine such as the values of the program counter and machine registers, just before the procedure is called.
4. The optional access link is used to refer to non local data held in other activation records.
5. The optional control link points to the activation record of the caller.

6. the field for actual parameters is used by the calling procedure to supply parameters to the called procedure.

7. the field for the returned value is used by the called procedure to return a value to the calling procedure.

the sizes of each of these fields can be determined at the time a procedure is called.

STORAGE - ALLOCATION STRATEGIES

A different storage - allocation strategy is used in each of the three data areas, in the organisation,

→ static allocation
→ stack allocation
→ Heap allocation

ATIC ALLOCATION

In static allocation, all programmer defined variables are bound to storage locations as their declarations are processed.

Temporary variables, including the one used to save the return address, are also assigned fixed addresses within the program. So everytime a procedure is activated, its name are bound to the same storage locations. this property allows the values of local names to be retained across activations of a procedure.

The compiler determines the amount of storage required for a name from its type. The compiler must eventually decide where the activation records go, relative to the target code.

Once this decision is made, it is possible to fill in the addresses at which the target code can find the data it operates on at the compile time.

Also, we can find the addresses at which information is to be saved when a procedure call occurs.

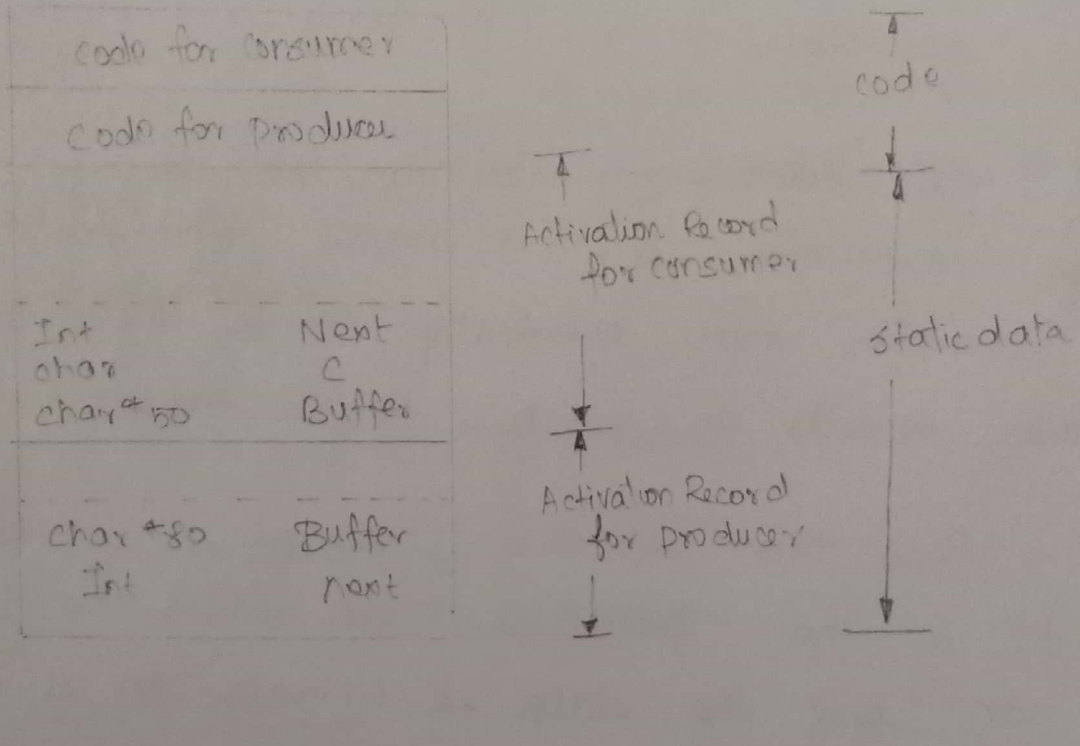
The limitations of the static memory allo

are :

1) It doesn't support recursive procedure.

2) the size of the data objects must be known during compilation.

3) As there is no mechanism for storage allocation at run time, data structures can be created dynamically.



allo

STACK ALLOCATION

Stack Allocation is based on the idea of a control stack. Storage is organised as a stack, and activation records are pushed and popped as activations begin and end, respectively.

For each call of a procedure, activation record contains storage for the local variables of that call. these values of locals are lost when an activation ends, as its activation record is popped off.

the top of the stack may be stored in a register. At runtime, an activation record can be allocated and deallocated by incrementing and decrementing top, respectively, by the size of the record.

CALLING SEQUENCES

Procedure calls are implemented by generating calling sequences. All call sequence allocates an activation record and enters information into its fields. A return sequence restores the state of the machine so the calling procedure can continue execution.

the code in a calling sequence is Often

between the calling Procedure (the caller) and

Procedure it calls (the caller).

In the General activation record, the cont

link, access link and machine-status fields appa

in the middle. the decision about whether or

not to use control and access links is part of the

design of the compiler, so these fields can be

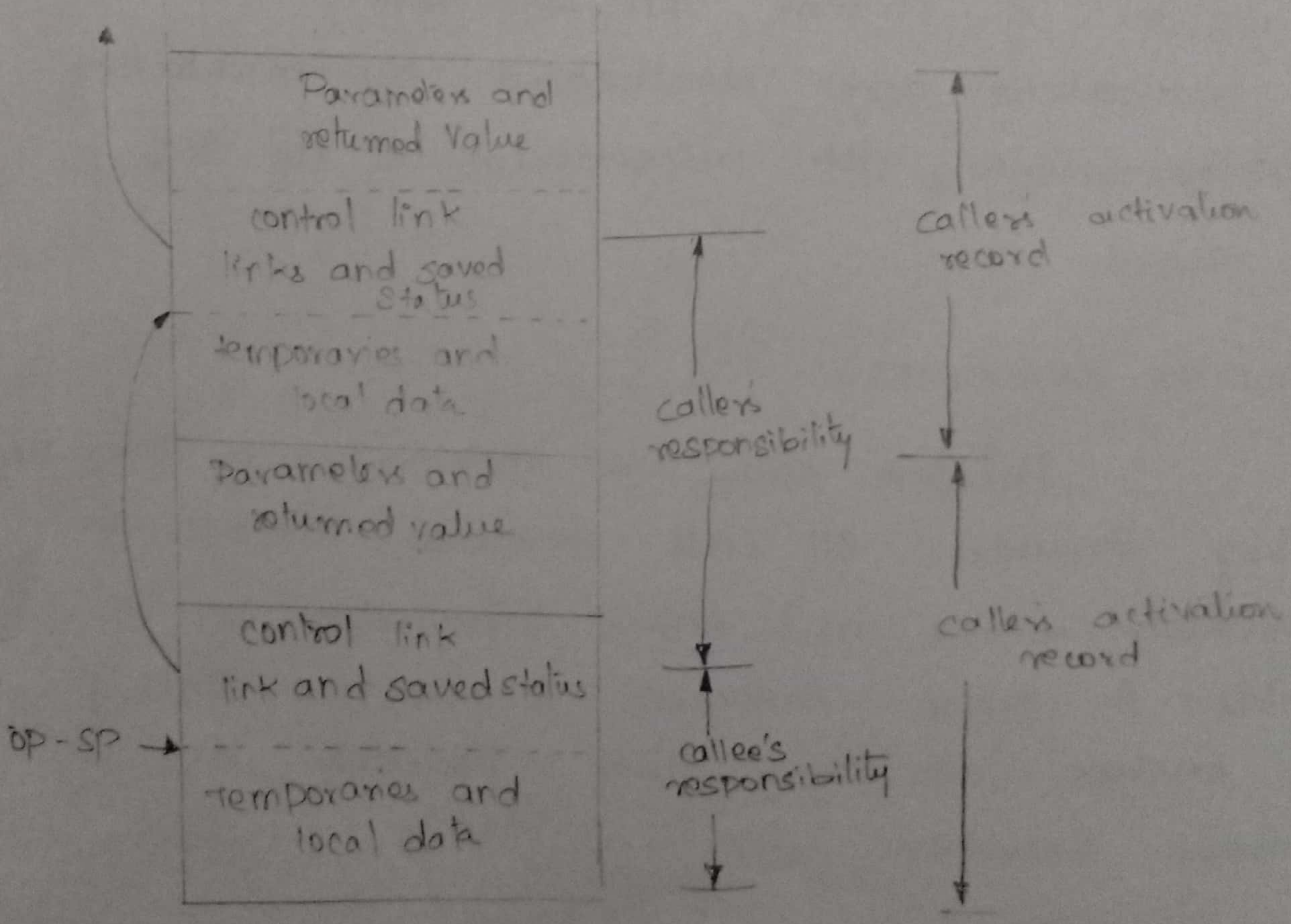
fixed at compiler - construction time. If exactly

the same amount of machine - status information

is saved for each activation, then the same code

can do the saving and restoring for all

activations



Parameters and returned value
control link
links and saved status
temporaries and local data
Parameters and returned value
control link
link and saved status
op-sp
temporaries and local data
callers activation record
callers responsibility
callers activation record
callee's responsibility

Even though the size of the field for temporaries is eventually fixed at compile time, this size may not be known to the front end. There this field is shown after that for local data, where changes in its size will not affect the offsets of data object relative to the fields in the middle.

The caller can access parameters and returned value fields using offsets from the end of its own activation record, without knowing the complete layout of the record for the callee, by placing these fields next to the activation record of the caller.

The register top-sp points to the end of the machine - status field in an activation record.

~~the register to~~

The code for the callee can access its temporaries and local data using offects from top-sp. The call sequence is:

1. The caller evaluates actuals.

2. The caller stores a return address and the old value of top sp into the caller's activation record. The caller then increments top-sp to the position.

Scanned by CamScanner

3. the callee saves register values and status information.

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

A possible return sequence is

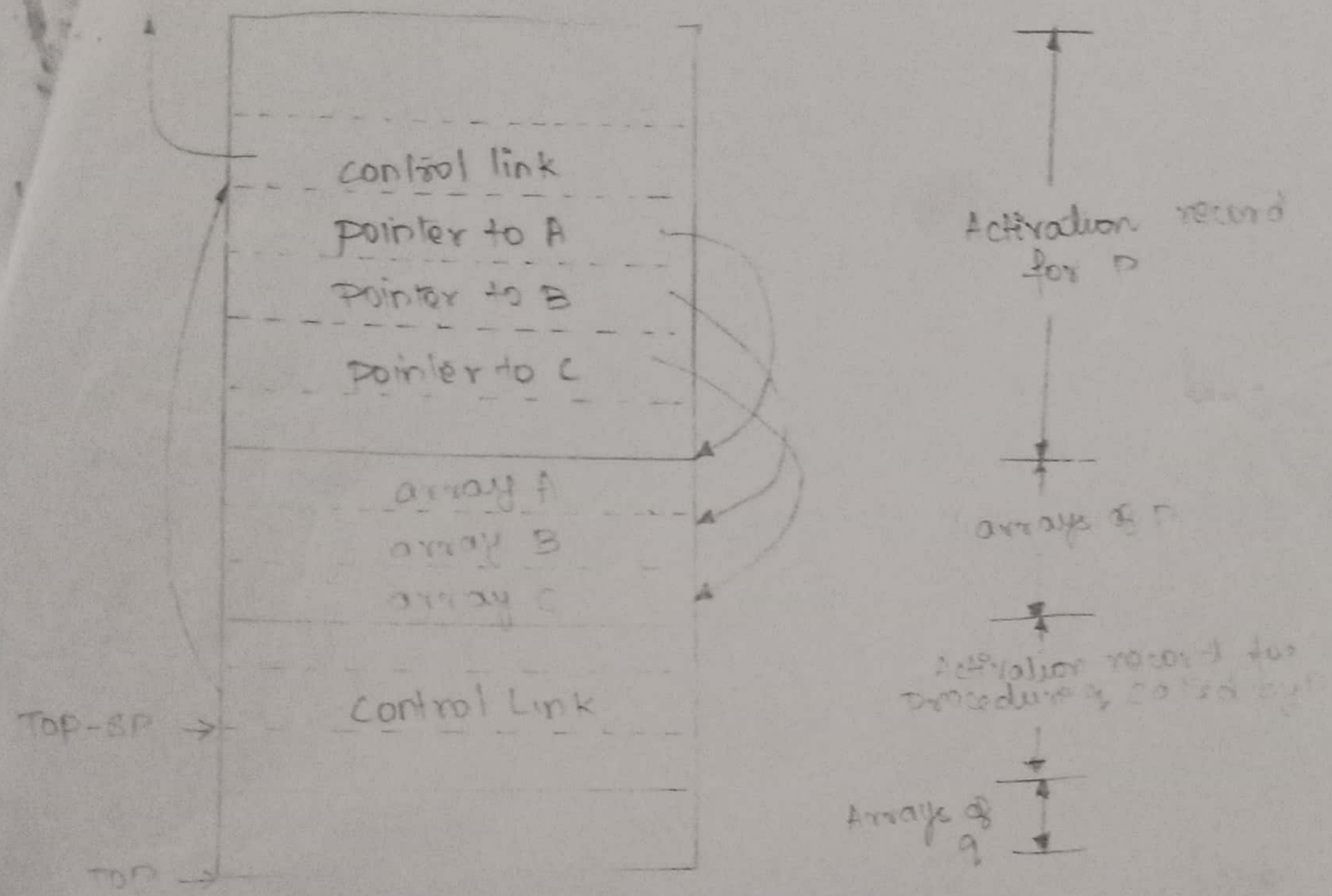
1. The callee places a return value next to the activation record of the caller.

2. Using the information in the status field, the callee restores top-sp and other registers and branches to a return address in the callers 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.

VARIABLE LENGTH DATA

the size of a local array may depend on the value of a parameter passed to the procedure. The size of all the data local to the procedure cannot be determined until the procedure is called. Only a pointer to the begining of an array appears in the activation record. the relative addresses of these pointers are known at compile time, so the target code can access array elements through the pointers



the activation record for q begins after the arrays of p, and the variable length arrays of q begin beyond that.

top points to the position at which the next activation record will begin. top-sp is used to find local data.

In figure, top-sp points to the end of the machine-status field in the activation record for q.

when q returns, the new value of top is top-sp minus the length of the machine-status and parameter's fields in q's activation record. After adjusting top, the new value of top-sp can be copied from the control link of q.

Whenever storage can be deallocated, the Problem of dangling references arises. A dangling reference occurs when there is a reference to Storage that has been deallocated.