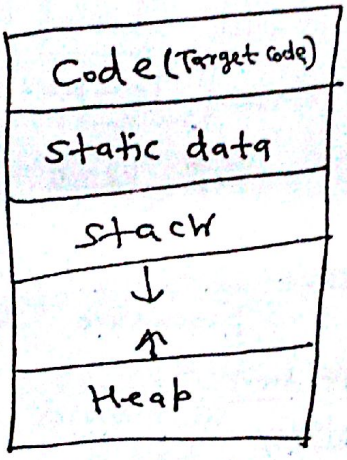
Run Time Storage ~~Allocation~~ Organization

- The Compiler obtains a block of storage from the operating system & use it for running the Compiled program. This block of memory is called runtime storage.

- The runtime storage is divided to hold
  (i) Generated Target Code
  (ii) Data object
  (iii) Control stack to keep track of procedure activations.

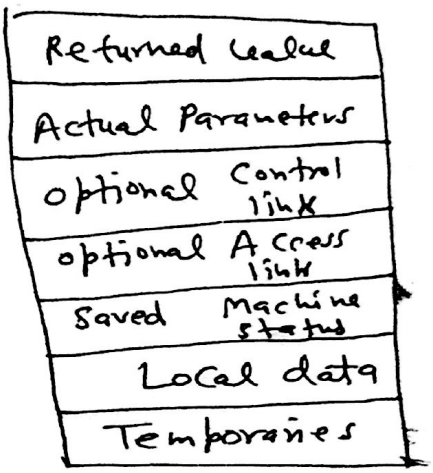


Runtime Storage organization

- The size of the generated target code is fixed at Compile time, so compiler place them in statically determined area. Similiary the size of some data objects are also known at Compile data, so they are also placed in statically determined area.
- Counter part of stacks is used to manage activation of procedures. When a procedure call occurs, execution of an active procedure is interrupted & status (Program Counter & value of machine registers) is saved on stack.
- A seperate area called heap holds all other information.

Activation Record:- Each execution of a procedure is called its activation.

- The activation record is a block of memory used for managing information needed by a single execution of Procedure.
- The various fields of activation records are as follows:-



General Activation Record

(i) **Returned value:-** This field is used by the called procedure to return a value to the calling procedure.

(ii) **Actual Parameter:-** This field is used by the ~~called procedure~~ calling procedure to supply parameters to the called procedure.

(iii) **optional control link:-** It points to the activation record of the calling procedure.

(iv) **optional Access Link:-** used to refer to non local data held in other activation records.

(V) **Saved Machine Status:-** It holds information about the state of the machine just before the procedure is called.

(VI) **Local Data:-** It holds data that is local to an execution of a procedure.

(VII) **Temporaries:-** Temporaries variables are needed during the evaluation of expressions.

Storage Allocation strategies:

There are three different storage allocation strategies:-

(1) **Static Allocation:-** * It allocates storage to all the data objects at compile time.

* The binding of name with the amount of storage allocated do not change at run time. Due to this, the name of this allocation is static allocation.
* In static allocation, the compiler can determine the amount of storage required by each data object.
* The limitations of static allocation are
  (a) The data structures can not be created dynamically.
  (b) Recursive procedures are not supported by this type of allocation.

(2) **Stack Allocation:-** * In stack allocation, stack is used to manage runtime storage.

* Data structures and data objects can be created dynamically.
* Using Last in first out (LIFO) activation records and data objects are pushed onto the stack.
* It supports dynamic memory allocation but it is slower than static allocation strategy.
* It supports recursive procedures.

(3) Heap Allocation:-

- In heap allocation, heap is used to manage dynamic memory allocation.
- Data structures & data objects can be created dynamically.
- A contiguous block of memory from heap is allocated for activation record or data object.
- The effecient heap management can be done by
  (a) creating a linked list for the free blocks.
  (b) Allocate the most suitable block of memory from the linked list.

Error Detection & Recovery:- Error detection & recovery is one of the imp. function performed by compiler.

Error Handling Process:- The process of locating errors and reporting to user is called "error handling process".

Function of Error Handler:-

(1) **Detection:-** The error handler should identify all possible errors from source code.

(2) **Reporting:-** It should report these errors with appropriate messages to the user.

(3) **Recovery:-** The error handler should recover from these error by repairing them in order to continue processing of the program.

(2) Syntatic Errors.

- The errors that occurs during the syntax analysis phase of the compiler is known as syntatic error.
- Syntatic errors are arises due to the missing operators, misspelled keywords and unbalanced parenthesis.
- The syntax analyzer receives tokens from lexical analyzer and if the token do not follow the grammatical rules of programming language, then the syntactical errors occurs.
- strategies for recovery from syntatic errors:-

(a) Panic Mode:- This is widely used technique. It is also simpler to implement. In this method, the parser discards input symbol one at a time & continue until one of designated set of synchronizing tokens is found. This method is mainly successful when there are less no. of errors in the same statement.

(b) Phrase Level Recovery:- In this method, on discovering error, parser perform local correction on remaining input. The local correction can be replacing comma by semicolon, deletion of extra semicolons or inserting missing semicolon. The only disadv. of this method is that, it find difficult to handle the situation, where the actual error has occured before the point of detection

(3) Semantic Errors:-

- Semantic errors are those errors which get detected during semantic analysis phase.
- The various errors in this phase are:-
  (a) incompatible types of operands.
  (b) Undeclared variables.
  (c) Not matching of actual arguments with formal arguments.
- If the data types of two operands in an expressions are not compatible with each other, then automatic type conversion is done by compiler.

Storage allocation for block structured Language:

In block structured language, the storage allocation can be done for two types of data variables.

(1) Local data (within the same procedure) (2) Non local data. (outside a given procedure)

(1) Local Data:- The local data can be accessed with the help of activation record. The offset relative to base pointer of an activation record points to local data variables within an activation record.

(2) Access to Non Local data:- A procedure sometimes refer to variables which are non local to it. Such variables are called Non local variables.

The way that the nonlocals are accessed depends upon the scope rules of the language.

The scope rules determine the treatment of references to non local names.

There are two different types of scope rules for non local names.

① Static scope Rules ② Dynamic scope rules

① **Static scope Rules**:- ✱ also called Lexical scope rules

- ✱ determined at Compile time.
- ✱ Determines the declaration that applies to a name by examining the program text alone.
- ✱ Pascal, C & Ada languages uses static scope rule.
- ✱ This is used by block structured languages.

② **Dynamic scope Rules**:-

- ✱ determines the declaration applicable to a name at run time, by Considering the current activation record.
- ✱ Lisp, APL & Snobol are among the languages that use dynamic scope.
- ✱ They are used by non block structured language.

Access Links & Displays to access Non-local names:-

A procedure may access non local names using either access link or displays

Access links:- ✱ A direct implementation of lexical (static) scope for nested procedures is obtained by adding a pointer Called "Access link" to each activation record.

✱ If procedure p is nested immediately within q in the source text, then the access link in an activation record for p points to an access link in the record for the most recent activation of q.

The Code to set up access link can be determined at ~~run time~~ Compile time, using the idea of nesting depth.

(a) The outmost scope has nesting depth = 1

(b) The nesting depth increses by 1 each time we enter a new scope & decreses by 1, when we leave a scope.

If procedure A at depth n_A calls procedure B at depth n_B then

(i) If $n_A < n_B$, then B is enclosed in A and $n_A = n_B + 1$

(ii) If $n_A \geq n_B$, then it is either a recursive call or calling a previously declared procedure.

Displays:- The purpose of using displays is to avoid the runtime overhead of multiple access - links to find the activation record that contains a non-local usable. The display may be stored in register or in memory.

display is an array of pointers to activation records used to access activation record.

In display:-

(a) An array of pointer to activation record is maintained.

(b) Array is indexed by nesting level.

(c) The pointers point to only accessible activation record.

(d) The display changes when a new activation occurs & it must be reset when control return from the new activation

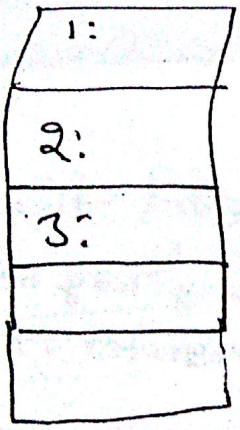
(e) It is an effecient way to access nonlocal names.

→ block

Comparison b/w Access links & Displays

1. Access links can require more time (at runtime) to access non-locals.
2. Display require more space at runtime.
3. Display is used to generate code effectively.
4. It can also require more time to setup a new access link when a procedure is called.

→→ In display, for each level, there will be an array entry



current activation record at Level 1.

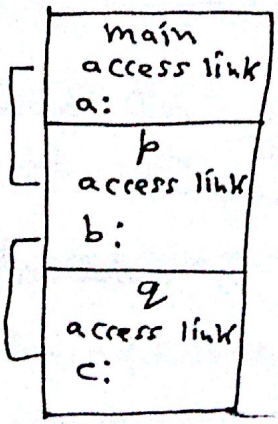
Current activation record at level 2

Current activation record at level 3

Example for Accessing Non Local data using Access links & displays:

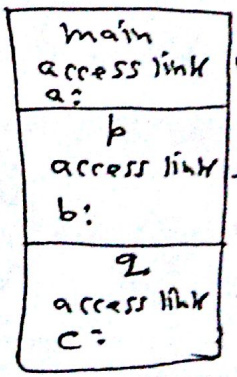
```
Program main();
 var a : int;
 Procedure p();
  var b: int;
   begin z; end.
 Procedure q();
 var c: int;
 begin
    c: a+b;
  end;
begin p; end.
```

Using Access link




```
addrC := offset C(currAR)
   t := * currAR
addrB := offsetB(t)
   t := *t
addrA := offsetA(t)
ADD addrA, addrB, addrC
```

Using Display




```
addr C := offset C(D[3])
addr B := offset B(D[2])
addr A := offset A(D[1])
ADD addrA, addrB, addrC
```