6/11/2023

UNIT-4

Symbol Table 2D Runtime Storage Environment

→ Runtime environment ~~is~~ means when you run the program, what is the support that you want from the operating system.

"RTE" (Run Time Environment) is how a program should loaded in the operating system and what is the support that your program needs.

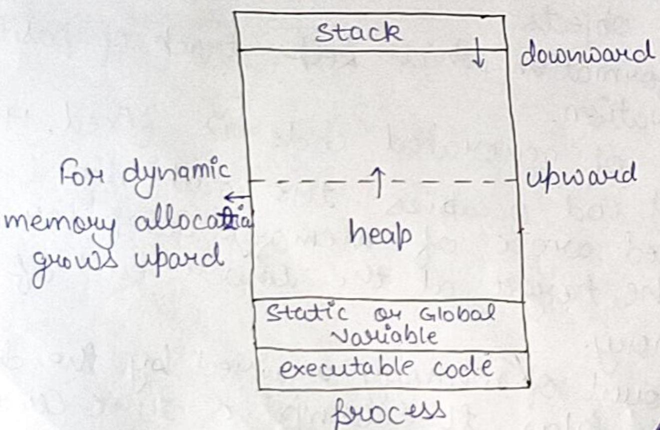
- The compiler demands for a block of memory to the OS. The compiler utilizes this block of memory for running (executing) the compiled program. This block of memory is called run time storage.
- The runtime storage is sub-divide to hold code & data such as—
  - i) The generated target code.
  - ii) Data objects.
  - iii) Information which keeps track of position activation.
- The size of generated code is fixed. Hence the target cod occupies the statically determined area of memory. Compiler places the target at the lower end of the memory.
- The amount of memory required by the data objects is klas the compile time and hence data objects can also be placed at the statically determined area.

→ The counter part of control stack is used to manage the active procedures. Managing of active procedures means that when a call occurs then ~~activation~~ execution of activation is interrupted and information about status of the stack is same on the stack. When the control returns from the call this suspended activation is resumed after storing the value of the element registers.

{Static or global variable
{executable code.

→ The heap area is the area of run time storage in which other information is stored. For example memory for some data items is allocated under the program control.

→ The size of a stack and heap is not fixed ~~if~~ it may grow or shrink interchangeably during the program execution



→ For a process space is provided by OS. Every OS provides three types of storage technique

i) Static
ii) Stack
iii) Heap

Storage Allocation Strategy

① **Static** :- i) Allocation is done at compile time.

ii) Bindings do not change at run time.

iii) One activation record per procedure.

Disadvantage

i) Recursion is not supported.

ii) Size of data objects must be known at compile time.

iii) Data structures can not be created dynamically.

② **Stack** : i) When a new activation begin activation records is pushed onto the stack and whenever activation ends, activation record is popped off.

ii) Local Variable are bound to a fresh storage.

Disadvantage

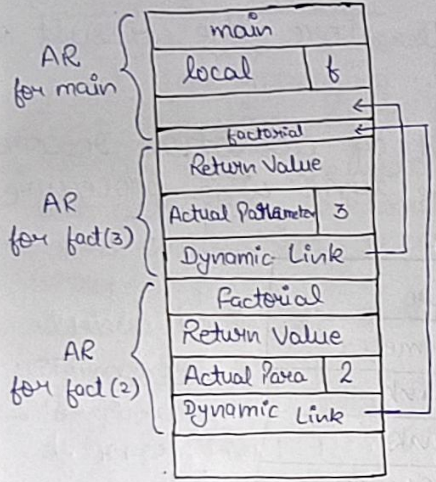
i) Local Variable can not be detained once activation ends.

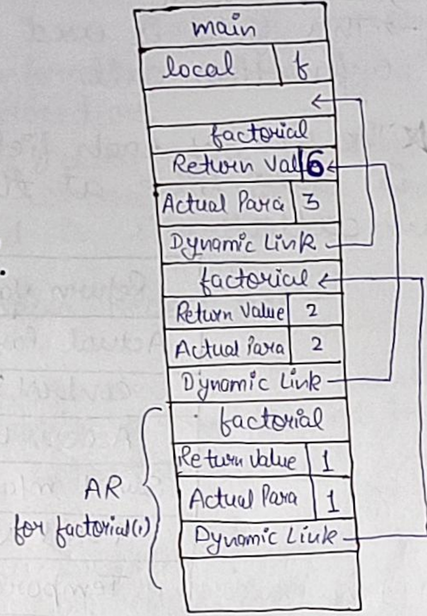
③ **Heap** :- Allocation and deallocation can be done in any order.

Disadvantage

→ Heap management is over-head.

Summary





Third Cell

Block Structure and Non-Block Structure Storage Allocation.

→ The storage allocation can be done for 2 types of data variables
- i) Local data
- ii) Non Local data.

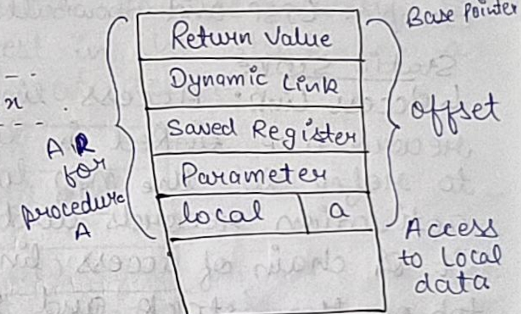
⇒ The local data can be handled using activation record whereas non local data can be handled using scope information. The block structured storage allocation can be done using static scope or lexical scope and the non-block structured storage allocation can be done using dynamic scope.

i) 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. Hence,

Reference to any variable x in procedure = Base pointer pointing to start of procedure + offset of variable x from base pointer

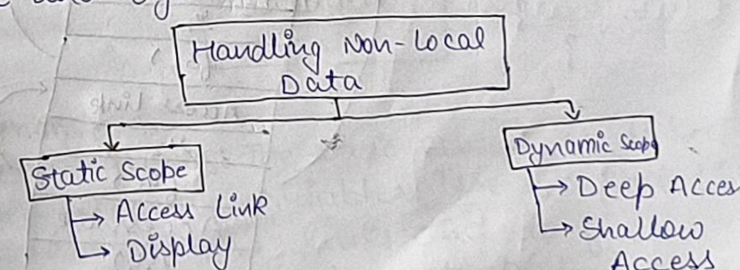
For example:

```
a()
{
------
  int x.
} -------
```



Access to non-local data

→ A procedure may sometime refers to variables which are not local to it. Such variables are called non-local variables. For non-local names there are two types of scope rules static and dynamic.



Static Scope Rule

→ In this type the scope is determine by examining the program text.
Example: Pascal, C, ADA

Dynamic Scope Rule

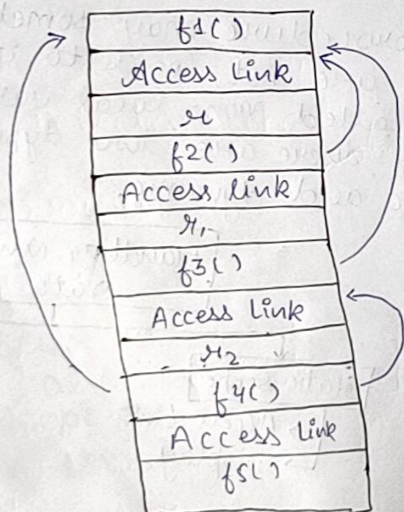
→ The dynamic scope rule determines the scope of declaration of the names at runtime by considering the current activation.
Example: LISP and SnowBall.

Static Scope

↳ Access Link: Access links in one activation record are ~~linked in~~ links that are used to refer to the non local data in other activation records and provide access to it. A chain of access link is formed on the top of the stack and represents the program scopes or static structure

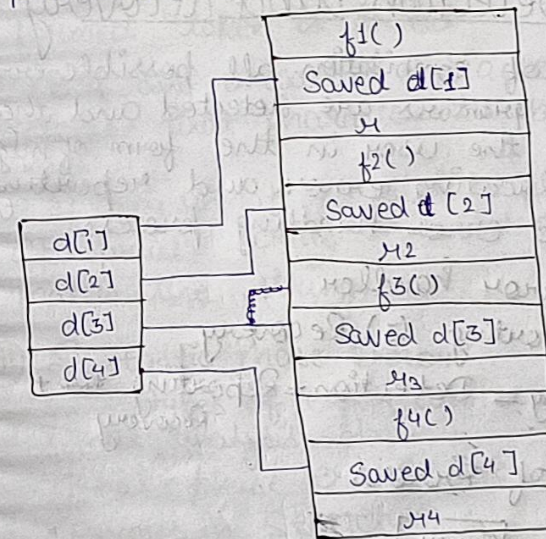
Ex:

```
f1()
{
   f2();
   f3();
   { f4();
   }
}
```



One of the issues of the access links is that if there are many nested calls of functions there will be a very long chain of access link that we must follow to reach the non local data we need. To solve this we make the use of display.

Displays are auxilary arrays that contain a pointer for every nesting depth. At any point d[i] is the pointer to the activation record which is the highest in the stack for a procedure which is at depth i



Dynamic

(i) Deep Access: The basic concept is to keep a stack of active variables. Use control links instead of access links and to find a variable, search the stack from to to bottom looking for the most recent activation record that contains the space for desired variables. Since search is made deep in the stack hence the method is called deep access.

(ii) Shallow access: The idea to keep central storage and allot one slot for every variable. If the range is not created at run-time then the storage layout can be fixed at compile time.

Error Detection and Recovery

→ In the process of compilation all possible errors made by the programmers are detected and they are reported to the user in the form of msgs. This process of locating errors and reporting to user is called error handling process.

Functions of Error handler

1) Detection (2) Reporting (3) Recovery

Error handling = Detection + Reporting + Recovery

Classification of Errors

- Error
  - Compile time error
    - Lexical Phase Error
    - Syntactic error
    - Semantic Error
  - Run time error

(i) Lexical Phase Error: These types of error can be detected during lexical analysis phase. Typical lexical phase errors are —

- Exceeding length of identifier or numeric constant.
- Appearance of illegal characters.
- Unmatch string.

↳ Error Recovery

→ Panic mode error recovery: In this recovery mechanism successive characters from the remaining i/p are deleted until the well formed token is found

→ If any unwanted character occurs then delete that character to recover from the error

→ If any unmatched string occurs then insert appropriate string or character in order to match the string.

(ii) Syntactic Phase Error.

→ These types of error can be detected during syntax analysis phase.

Typical syntax phase errors are

- Errors in Structure
- Missing operator
- Miss-spelled keywords
- Unbalanced analysis
- Unbalanced paranthesis

Strategies

Strategies to recover from syntatical error-

⇒ Parser employes various strategies to recover from syntactic error. These errors are -

i) Panic mode
ii) Phrase level Recover
iii) Error Production
iv) Global Correction.

(iii) Semantic Error.

→ Semantic errors are those errors which get detected during semantic analysis phase.

Typical errors are:

* Incompatible types of operands
* Undeclared variables.
* Not matching of actual arguments with formal arguments.

Ex:

```
int a[10], b
a=b;
```