compilers & Interpreters.

- Aspects of compilation -

What is meant by compilation → compilation is done by compiler.

- What is compiler.
- A compiler bridges the semantic gap between a PL domain & execution domain.

- Two aspects of compilation are:
- i) generate code to implement meaning of source program in the execution domain. of source program in the execution domain.
- ii) Provide diagnostic for violation of PL semantics in a source program.

features are:
- Data types
- Data structures
- Scope rules
- control structure.

i) Data types :-
- A data types is the specification of
1- legal values for variables of the type &
2- legal operations on the legal values of the type.

- Following tasks are involved.
1- checking legality of an operation for the type of its operands. This ensures that a variable is subjected only to legal operations of its type.

2) - use type conversion operation to convert value of one type into values of another type wherever necessary & permissible according to the rules of a PL.

e.g - Handling data types.

- consider the C program.

```
Float x, y
int i, j;
y = 10;
x = y+1 ;
```

- While compiling 1st assignment statement.
- The compiler must note that y is of type of float.
  ∴ - it must generate code to convert the value 10 the the floating point.

In the second assignment statement.

- The value of y & i cannot add straightway because they are of different types.

Hence - the compiler must first generate code to convert the value of i to the floating point & then generate code to perform the addition as a floating operation.

result. y is floating value.

so compiler convert float value. like

y = 10

y = 10.00.

result is

x = y+1

x = 10.00 + 1

= 11.

(Handling a user-defined data type).

```
Program - user-defined data (input, output);
type
    weekday = (mon, tue, wed, thu, fri);
var
    today : weekday;
begin { main Program }
    today := mon;
    if today = tue then ...
```

- Here weekday is a user-defined data type
- The compiler first decide how to represent the values of the type - map the values into subrange of int.
  e.g. values mon ---- fri - mapped into subrange 1 ... 5.

value → mon tue wed thu fri
integer → 1 2 3 4 5 like these.

- This representation does not cause a confusion between days of the week & integer.
  — because the compiler coould have check legality of each operation in the program before generating code.

Thus, use of an expression like tue+10 in the program would not lead to the value 12. (2+10) = 12
— if tue is represented value 12, + is not legal operation of type weekday.

- so compiler would indicate an error & prevent the program reaching execution.

3) Use appropriate instruction seq - of the target machine to implement the operation of type.

4) Data Structure :-

- A programming language (PL) permits use of data structure arrays, stack, records, lists etc.
- To compile a reference to an element a data structure, the compiler must develop a memory mapping to access the memory allocated to the element.
- A record which is heterogeneous DS leads to complex memory mapping.

Memory mapping for accessing data structure

```
type
    employee = record .
      name : array [1...10] of character;
      id : integer .
    end;
var
    info : array [1... 500] of employee;
    i, j : integer;
begin { main program }
    . info [i] . id := j;
end.
```

Here - info is an arry of records . - info[i].id - involve use of two diff. kinds of mapping - The first mapping is used to locate the ref. element of the array . i.e info[i]. - it involve, multiply i-1 by size of element of info. data-type employee & adding result to address of memory emp + Am - allocated to info[1].

- 2nd mapping - use to access the field id of an emp record.

Scope rule.

```
Block A [ x, y : real.
          Block B [ i, j : integer;
                    x := y;
```

- First rule - those variable assign / declare particular block. this variable access of particular block.
  i.e - x,y assign Block A so. x,y variable access Block A.

- second rule - outer block access inner block.
  i.e - mean outer block variables access inner block.
  i.e - xy access Block B.

```
Block A [ x, y : real.
          Block B [ y, z : integer.
                    x := y;
```

- Variable x of block A is accessible in block A & also in the enclosed block B.
- However variable y of block A is not accessible in block B because block B contains a declaration of another variable, named y.
  Thus the statement x := y uses variable y of Block B & var. x of block A.
  variable y & z of block B are not accessible in Block A.

- A compiler performs operations called scope analysis and name resolution to determine which data item is designated by the use of a specific name at a specific place in the source program.

- The generated code simply implements results of these analyses.

iv) Control Structure :-
(Handling control structures).
While compiling the for statement in the program.

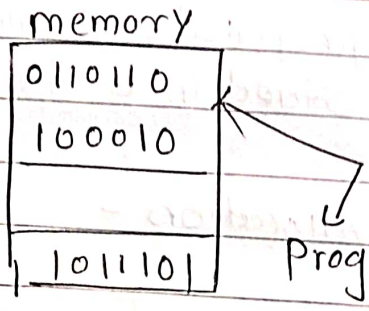
```
for i := 1 to 100 do
begin
    lab1: if i = 10 then...
end;
```

compiler has to generate code the value i from 1 & 100 & execute the if statement for each value i

- semantics of the for statement.
two restriction - A control transfer to the statement lab1 - from outside the loop is forbidden. so. assignment of value of is is also forbidden.
- The compiler has to check whether the source program satisfies these restrictions & indicate errors. When it does not.

◉ Memory Allocation.

- memory allocation is primarily hardware operation i.e - conducted or manage by operating system.



- Method of assigning memory to a process ~~a p~~ of element such variable instructions that need to be store in main memory in order to get process executed.

Process of allocating physical or virtual memory to process.

physical memory

It is an actual RAM of the system

virtual memory

It is an imaginary memory created in conjunction with hardware to extend the address space.

→ also known as physical memory.

Memory Allocation

- Static Allocation.
- Dynamic Allocation.

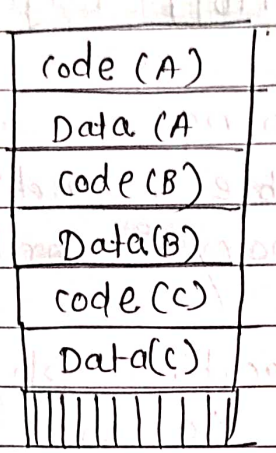
Memory allocation involve the following three tasks—

1- Determining the amount of memory required for storing the value of a data item.

2- using an appropriate memory allocation model for implementing the lifetimes of data items & scope rules of the language.

3- Developing appropriate memory mappings for accessing values stored in a data structure.

- Static Memory Allocation -



1) - Static memory Allocation is made when a program is compiled.

2) - The size required must be known beforehand. and thus it allots a fixed size.
that means- program must specified size required for particular element in program itself.

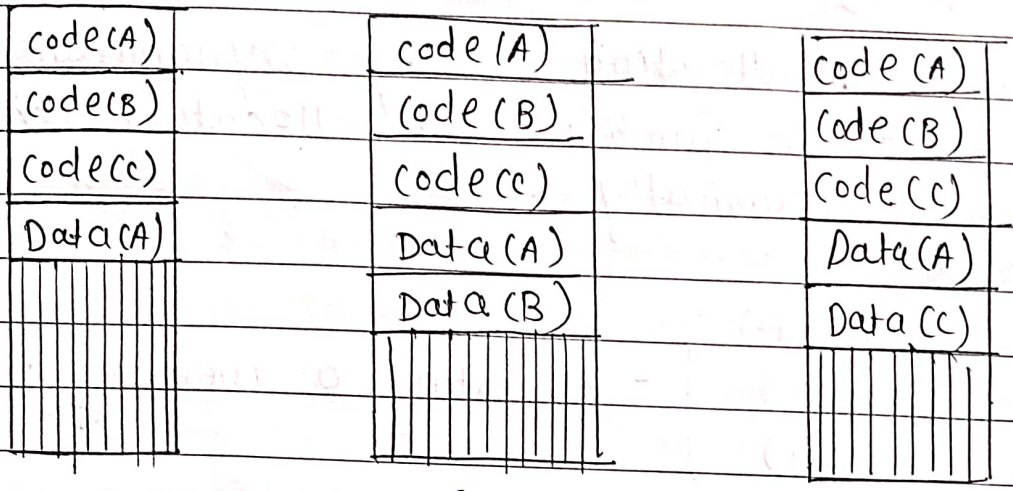
3) - If data size is not known then the compiler will assign a random size.
- with random memory size with occure two issue
i) - If the allocated size come out to be larger will lead to wastage.
ii) - If the allocated size is smaller it will lead to inappropriate execution.

4) - If the ~~allocated size~~

Advantages of static memory allocation
- It provide fast & efficient execution of the process.

Disadvantages:- you can not increase or decrease the fixed size memory allotted during static memory allocation.

- Dynamic memory Allocation.



Dynamic memory allocation

1) Dynamic memory allocation is conducted during the execution time.

2) - It allocates memory to program elements when they are used first time when program is running.

3) - Dynamic memory allocation takes place in heap memory.

4) - Exact amount of memory is allocated to program elements.

5) It makes execution flexible & improve system performance.

Advantages -

- Allocate the exact memory required.
- Reduce memory wastage.

- makes execution flexible.
- Improve the performance of the system.

disadvantage:
- Increases overhead of operanting system
- slowing down execution.

function of allocating memory Dynamically.
— Built-in functions for allocating or deallocating memory dynamically.

malloc() }
calloc() } - Allocating a memory.
realloc() }

free() } - deallocating a memory.

- We use pointers for accessing the memory that are allocated dynamically.

⑥ • storage Allocation in Block structured language -

```
Block A [ x,y : real;
Block B   [ y,z : integer;
            x := y;
```

— Diagram)

floatptr - means floating pointer.

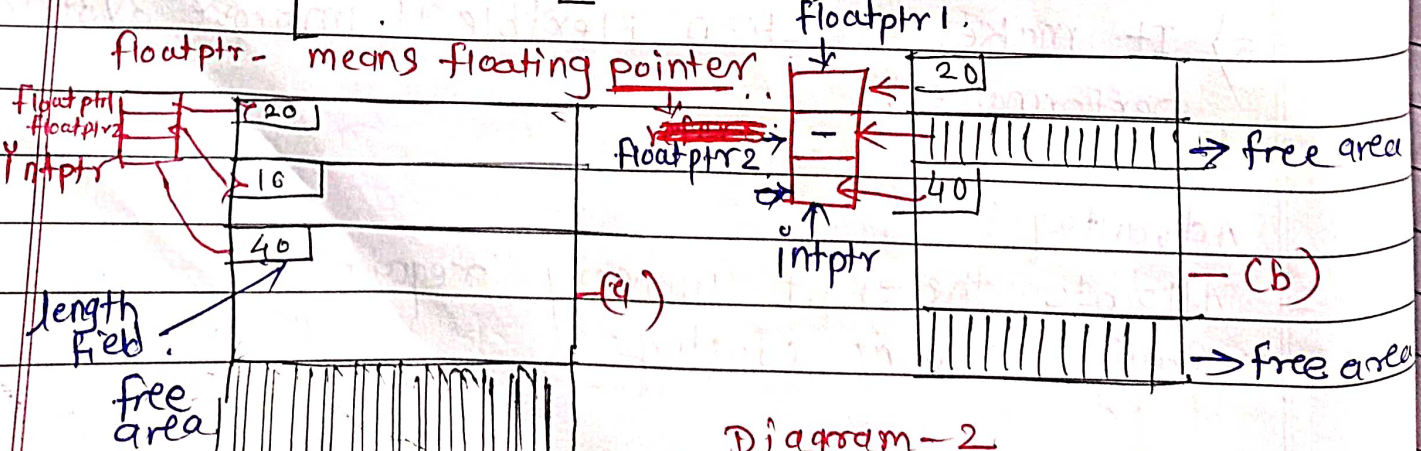
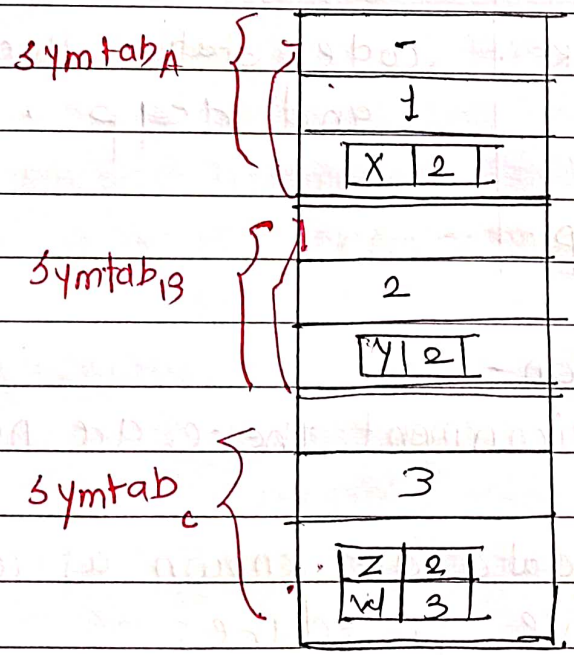


Diagram-2

(a-) heap
(b)- A hole in the allocation when memory is deallocated.

compiler performing following task.
1- Determine the static nesting level of
2- Determine the varible vari destinated by the name name; in accordance with the scope rules.
3- Determine the static nesting level of the block in which name; is defined. Determine the value of dvari in the activation record of the block.
4- Generate the code indicated in diagram. 4 dia-2

symbol table for a block structured language.

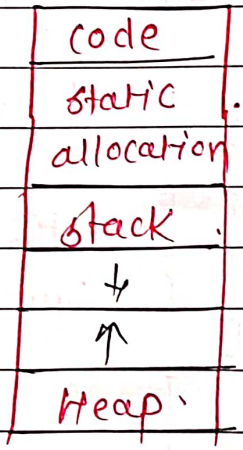


This activation record of a block contains the block's static nesting level & its symbol table.
- each entry in the symbol table show only symbol & its displacement in the activation record.
- The search for variable X accessed in the statement X := Z; of block C terminates on finding the entry of X in the symbol table for block A.

- The static nesting level of A is 1. dx = 2 if static nesting level of C, the current block.
- This information is adequate for generating code.

Run time Storage Allocation strategies.

- Static allocation
- Dynamic allocation
  - Stack allocation
  - Heap Allocation.



- Divide into 4 parts. code, static allocation, stack and Heap.

static Allocation -

1) In static environment there are number of restrictions:
   a) - size of data are known as compile time
   b) No recursive procedures
   c) No dynamic memory allocation.

2) only one copy of each procedure activation record exist.
   a) - we can allocate storage of compile time
   b) - Bindings do not change at runtime.
   c) - Every time a procedure is called, the bindings occurs.

- The storage is organised as stack.
- Activation records are push & pop.
- Activation record contains locals so that they are bound to fresh storage in each activation record.
- The value of locals is deleted when the activation ends.
- supports recurssive procedure.

Heap allocation.
- variables local to a procedure are allocated & de-allocated only at runtime.
- Dynamically allocate memory to variables claim it back when the variable are no more required.
- support recursive.

• Compilation of Expression •

- During code generation for expressions, the compiler has to ensure correctness of the generated code and its execution efficiency.
  - Major issues in code generation for expressions
1) - determination of the order in which operators used in an expression should be evaluate.

2) choice of instruction for performing on operation.

3) use of CPU registers & handling or partial results in the generated code
- A modern computer provides different ways of performing operation.

- The compiler has to make a suitable choice from these alternatives depending on the type & length of the operands, & the addressibility of each operand.

1) Operand descriptors / Address descriptors.
   - telling location / address of name.
- An operand descriptor has 2 fields.
  1. Attributes -
     contains the subfields type and length.
  2. Addressability -
     specifies where the operand is located, & how it can be accessed.
     - it has two subfields.
     a) Addressibility code - this code takes the values 'M' - operand is in memory &
        'R' - operand is in register.
     other addressibilities such as address in a register ('AR') & address in memory ('AM') are also possible;

     b) Address - Address of a memory word or a CPU register.

- An operand descriptor is built for each id, constant & partial result.

- A descriptor would be built for an id or constant when it is encountered in the expression.
  encountered means - evaluating expressing.

2) Register descriptors -

A register descriptor fields -

1) status - contains the code free or occupied to indicate register status

2) operand descriptor # : If status = occupied, this field contains the descriptor # for the operand contained in the register.

- The code generator builds a register descriptor for every CPU register at the start of its operation & store all these descriptors in an array named Register_descriptor.

example- use of operand descriptors & register descriptors -

code is to be generated for the expression a * b. Assuming a, b to be a type integer which occupies 1 memory word, the code generator builds two descriptors.

a * b

	Attributes	Addressibility
operand_descriptor[1]	(int, 1)	M, addr(a)
operand_descriptor[2]	(int, 1)	M, addr(b)

The code generated now generates the instruction

```
MOVER   AREG, A
MULT    AREG, B
```

To perform the multiplication & builds the descriptor for the partial result.

	Attributes	Addressability	comments
perand_descriptor[3]	(int, 1)	R, addr(AREG)	Descriptor for a*b

It also updates the register descriptor for AREG

Status	operand descriptor#
occupied	#3

which indicates that register AREG contains the operand described by descriptor #3.

- **Triples -**

- The optimization of common subexpression elimination described later removes some of the evaluation of a subexpression from a program it doing so does not change the meaning & results of a program.

- Its application requires knowledge of whether a subexpression occurs many times in a program.

- The intermediate codes called triples & quadruples facilitate efficient implementation of this operation

- A triple represents the evaluation of an operator

- The intermediate code of a program consists of a sequence of triple.

triple format.

operator	operand 1	Operand 2

quad ruples

format.

operator	operand1	operand2	Result

- are not temperary location for holding partial results..
- They are simply names.
- some of these become temporary locations when common subexpression elimination is implemented.

e.g - $-(a*b)+(c*d+e)$

$t_1 = a * b$.
$t_2 = -t_1$
$t_3 = c * d$
$t_4 = t_3 + e$
$t_5 = t_2 + t_4$.

quadruples

	operator	operand1	operand2	result.
0	*	a	b	t_1
1	-	t_1		t_2
2	*	c	d	t_3
3	+	t_3	e	t_4
4	+	t_2	t_4	t_5

in triple remove result.

	operator	operand1	operand2
0	*	a	b
1	-	(0)	
2	*	c	d
3	+	(2)	e
4	+	(1)	(3)

index value

quadruples	triples
1) Easy move statement	1) do not move statement
2) use Extra space	2) useless space.