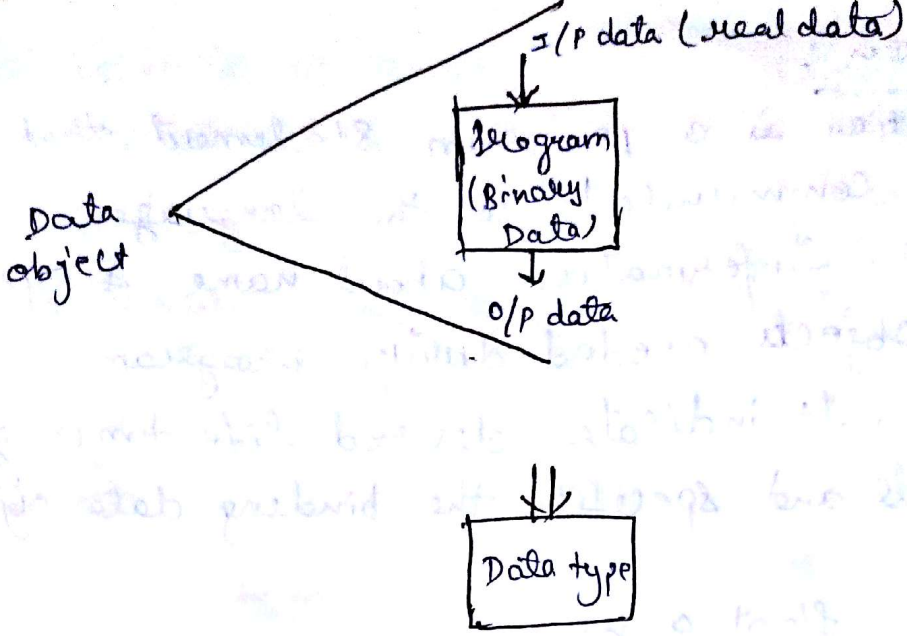
UNIT-II

1) Elementary Data Type.

Every program is a function that transforms a given set of inputs into an expected set of output. Set of input & output is known to the programmer and are called data. Input to program comes from real world which is then converted to binary form which makes the programmer lose connection with data of real world.

Programming relies on correct perception of relationship among various data items.
Type is category of entities. Data types are close to real world entities and hence make it easy for programming to focus on transformation to be done to produced desired set of output.
Compiler take care conversion of data type into machine recognizible form and vice-versa.



Advantages of Data type:

1. Error detection
2. Safety
3. Easy design
4. Abstraction
5. verification
6. Software evolution
7. Documentation

Properties of Data type:

1. **Data object**: It is runtime grouping of one or more piece of data in virtual computer. During execution of a program many different data objects of different type exists. Data object represents a container for data value. A place where data value may be stored.

2. **variable / Constant**: Variable is an elementary data object with a name.
Constant is a data object with a name i.e., bound to a value.

eg) int a;
int a=5;

(3) Declaration:

A declaration is a program statement that refers to communicate to the language translator. Information about name & type of data objects needed during program execution. It indicates desired life time of data objects and specifies the binding data object to names

eg) float a, b;

(4) Assignment: the rightmost 'x' refers to value contained in named data object (Rvalue) in the following example

```
X = X
```

the left most 'x' refers to location of data object that will contain the new value (L value)

i) Compute Lvalue of 1st operand expression
ii) Compute Rvalue of 2nd operand expression
iii) Assigned the Computed Rvalue to Computed Lvalue data object.
iv) Return the computed Rvalue as result of operation

(5) Initialization: An uninitialized data object is a data object that has been created but not yet assigned a value. Such variables are serious source of programming errors. The random bit pattern contained in

value storage area of an uninitialized data object cannot be distinguished from a valid value. So, immediate initialization of variable value on creation is often considered a good programming approach.

eg) int i=0;

(6) Type checking: It means checking that each operation executed by a program receives the proper no. of arguments of proper data types

eg) x=A+B*C; ← error.

i) Static type checking: It is performed during translation of a program. The needed information is usually provided in part by declaration with include -

- for each operation number, order & data type of its arguments & results
- type of data object associated with variable name.
- type of each constant data object

ii) Dynamic type checking: It is run time type checking usually performed immediately before execution of a particular operation. It is implemented by storing a "type" tag in each data object that indicates data type of the object. Operation is

performed only if the argument types are correct

Elements of Data type :

1) Attributes : Data type & name are attribute of any data type which are invariant during its lifetime.

2) value : there are all possible values the data type may contain.

3) Operation : It shows how data objects of that type may be manipulated

Implementation of Data type :

i) Directly as hardware operation.

ii) As a procedure or function subprogram

iii) As an inline code sequence.

Types of EDT :

1) Scalar Data type : they have a single attribute

eg → Numeric data type
int, float
→ characters
→ Booleans
→ Enumeration

2) Composite data types : /

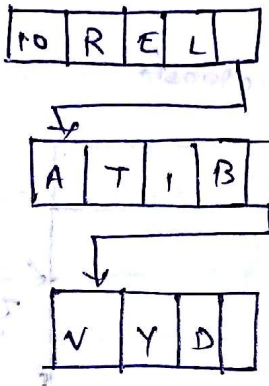
* Arrays : Collection of similar data type at contiguous memory location

a) Fixed length array: Data object may have a fixed length which is declared in the program

eg) int a[5];

b) variable length Array: It utilizes a descriptor containing both, maximum length & current length of string stored in data objects.

c) Unbounded length array: It is linked storage of fixed length data object which may be used to contain the string.



* Pointer: It defines a class of data object whose values are location of other data objects.

a) Absolute Address: Pointer value may be represented as the actual memory address of the storage block for the data object.

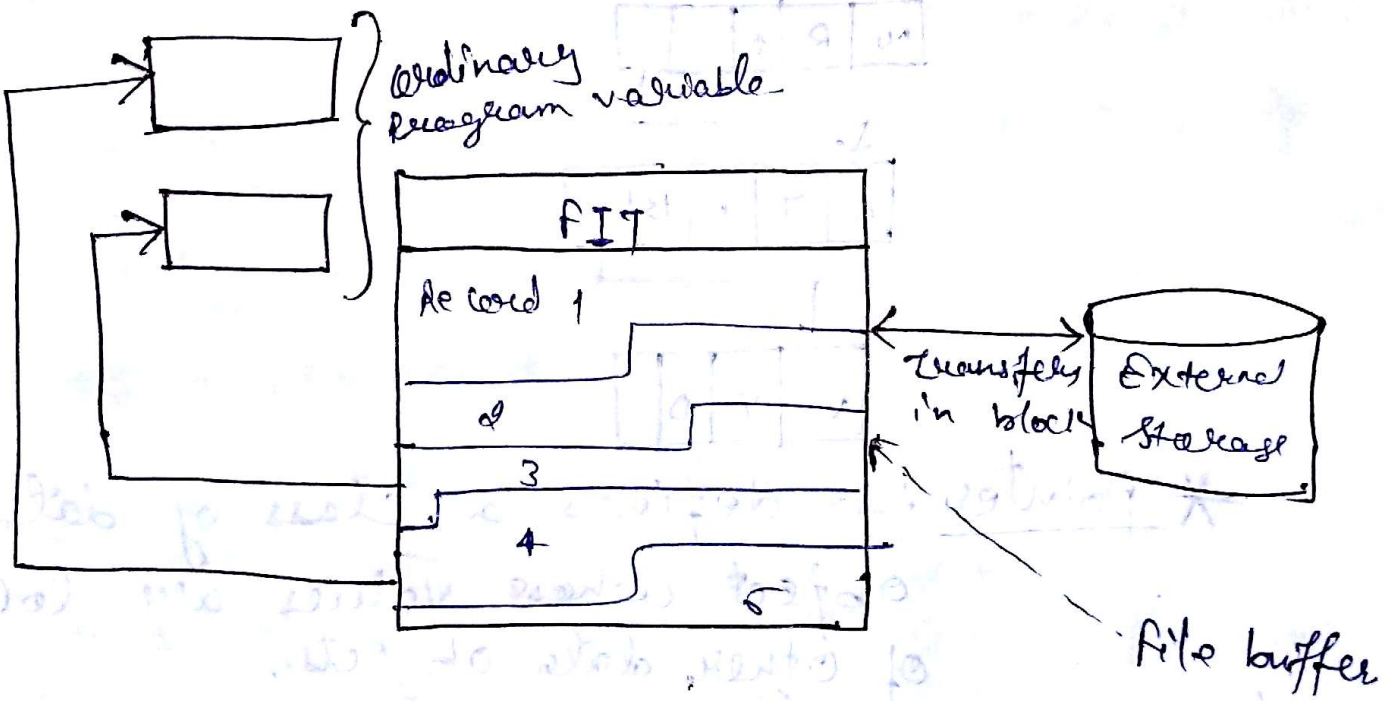
b) Relative address: Pointer value may be represented as an offset from the base ~~value~~ address of some large heap storage within which data object is allocated.

* file: Collection of record is called file. operations on files are read, write, etc.

when a program opens a file on file information Table (FIT) and buffer must be

operating system open & store information about location & characteristics of the file in FIT. the write primitive stores data in the next available position in buffer in the memory and updates the pointer to this buffer in FIT.

when file is read, each read operation executed by the program transfers the single component from buffer to program variable.



2.) Sequence Control

It is control of the order of execution of the operations both primitive & user-defined.

Types of Sequence Control:

1) Implicit Sequence Control: It is determined by the order of the statements in the source program by the built-in execution model of the language.

eg) expressions:

```
int a, b, c;
c = a + b;
```

2) Explicit Sequence control: Programmer uses statements to change the order of execution

eg) goto, if...else, break, continue, etc.

```
1) int a, b, c;
2) goto 6;
3) c = a + b
4)
5)
6)
```

Levels of Sequence Control:

1) Expression
2) Statements
3) Sub-programs
4) Declarative programming

1) Expressions: Properties like precedence rules and parenthesis determine how expressions are evaluated.

a) Syntax: How to make a tree and its various notations. like postfix, Infix, prefix.

b) Semantics: It includes precedence & associativity.

Precedence concerns the order of applying operation.

Associativity deals with order of operation of same precedence. Operators of same precedence are evaluated from left to right or right to left depending on the level. Thus associativity rule decides the order in which multiple occurance of same level operator are applied.

operand	Priority	Associativity
++, --	highest ↑	R to L
*, /, %		L to R
+, -		
<, <=, >, >=		
==, !=		
&&		
||	lowest	

2) Statements : Conditional & iteration statements determine how control flows from one segment of control to another.

* Composition :

Statements are executed in the order they appear in the source code

eg)
```
int a=2, b=3, c;
c=a+b;
printf("%d", c);
```

* Alteration :

Two sequences form alternatives so that one sequence or the other sequence is executed but not both.

eg)
```
if (a>b)
    printf("%d", a);
else
    printf("%d", b);
```

* Iteration :

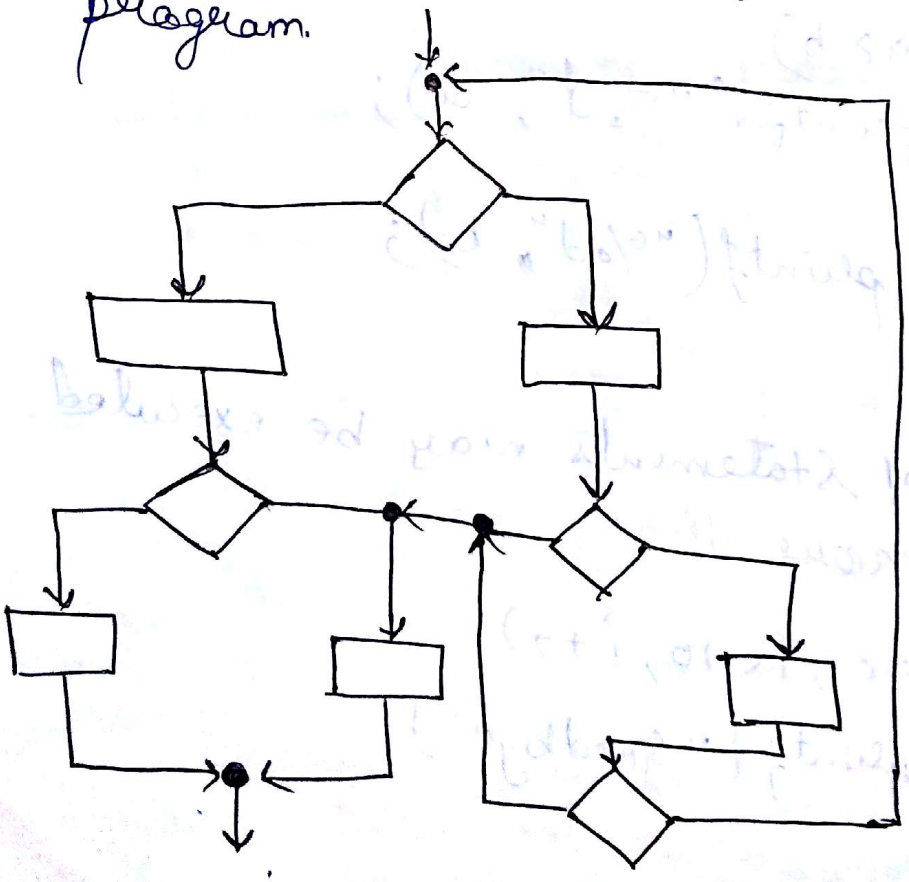
Sequence of statements may be executed zero or more times.

eg)
```
for (i=0; i<10; i++)
    printf("Goodbye");
```

Proper program: It is a formal model of a control structure as a flow chart that has three things —

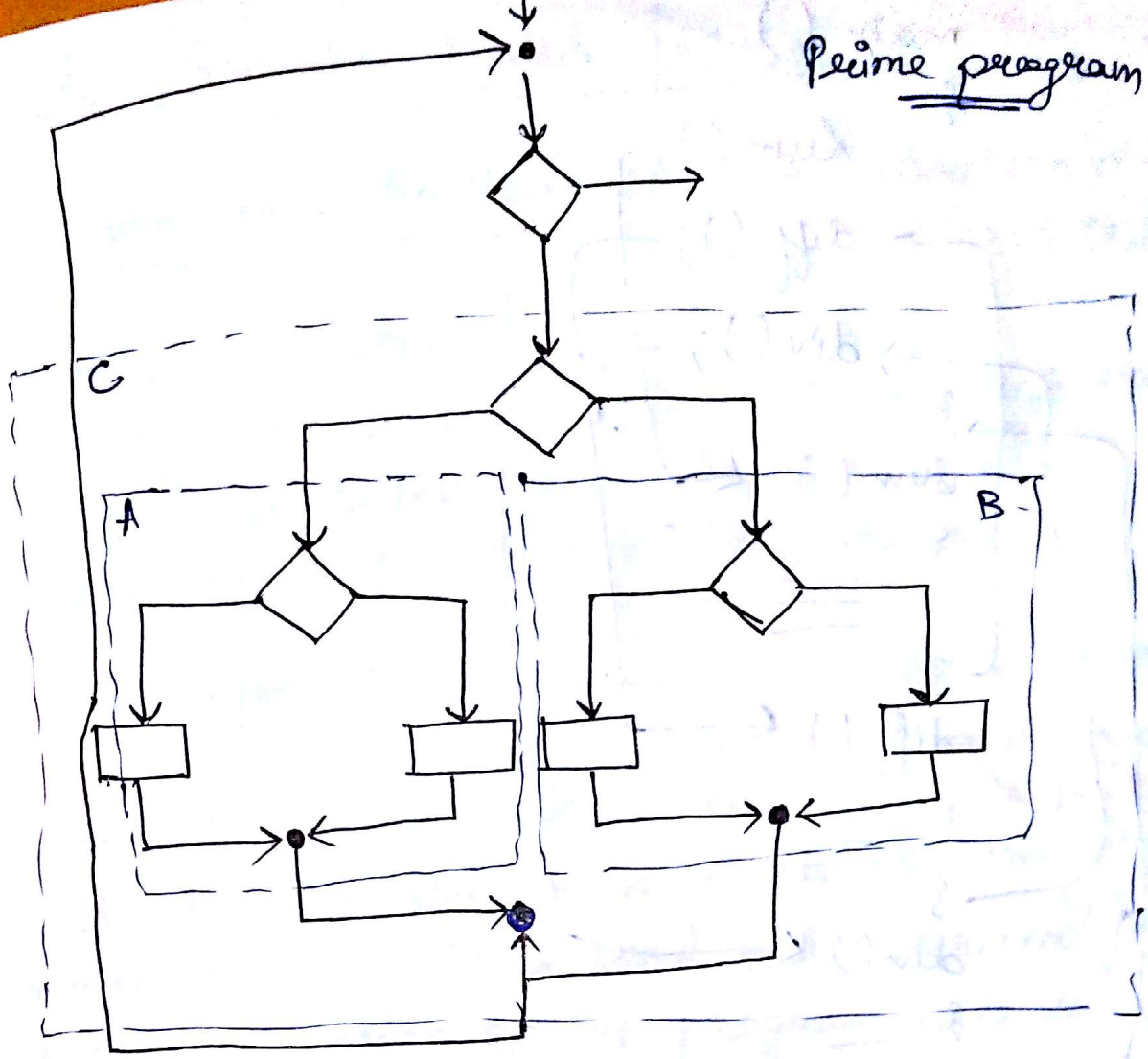
1) Single entry
2) Single exit arc
3) Has a path from entry arc to each node and from each node to exit arc

Prime program: It is a proper program that can not be sub-divided into smaller proper programs. If we can not cut two arcs of proper program to separate proper program into separate graphs then proper program is called a prime program.



Proper program

Prime program



3) Declarative Programming:

Execution model that does not depend on order of statements in some program.

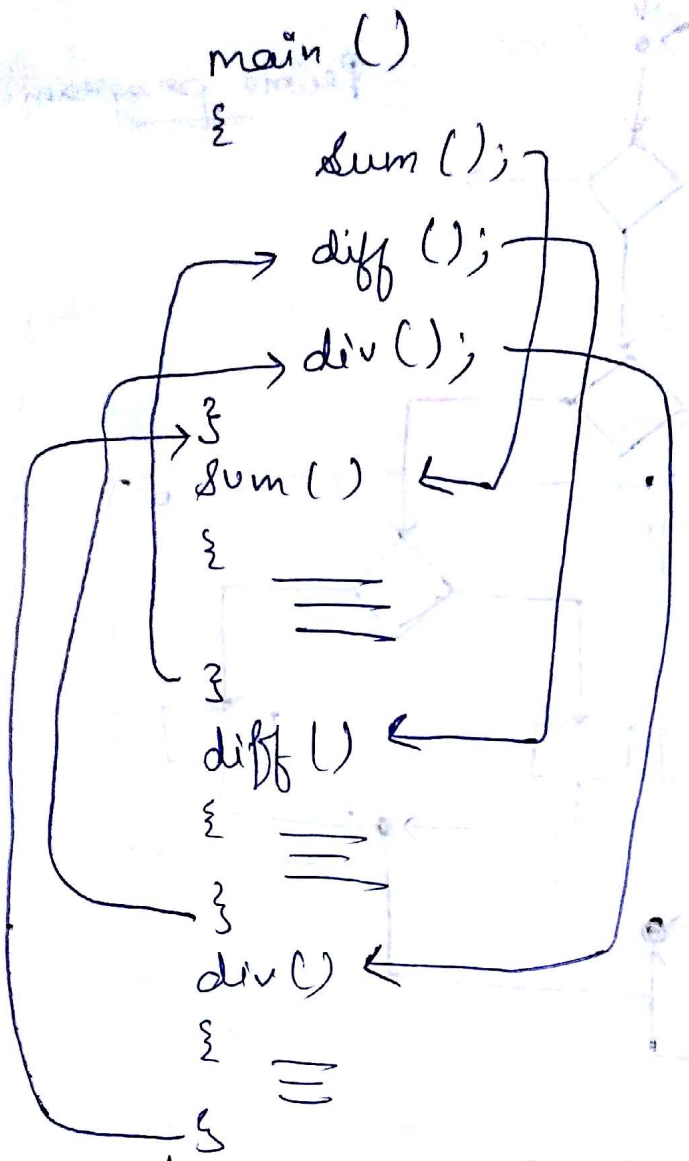
eg): John is parent of Mary.

parentof (John, Mary)

- Unification
- Resolution

4) Sub-program:

Transverse control from one program to another.



Copy rule :

A program is composed of a single main program which during execution may call various sub-programs which in-turn may each call other subprograms. Each sub-program at some point is expected to terminate its execution and return control to program that called it. During execution of a sub-program execution of the calling program is temporarily halted. When execution of sub-program is completed, execution of the calling program resumes at the point immediately following the call of the sub-programs.

Data Structure for implementation :

1) Sub-program definition: It is the written program which is translated into a template.

2) Activation: It is created each time a sub-program is called using template created from definition. It has two parts—

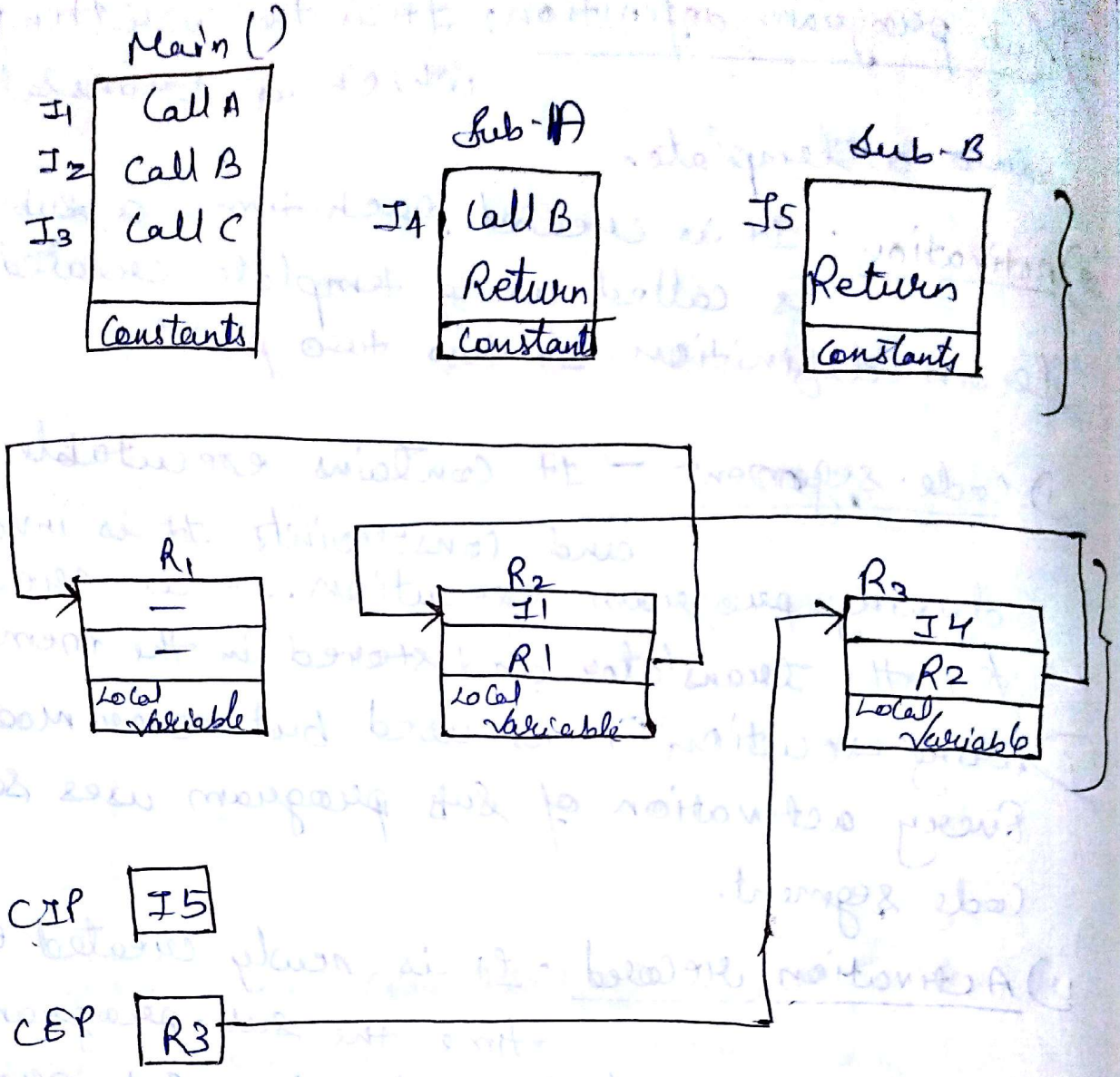
i) Code-segment — It contains executable code and constraints. It is invariant during program execution. It is created by the translator and stored in the memory. During execution it is used but never modified. Every activation of sub program uses same code segment.

ii) Activation record - It is newly created each time the sub-program is called and is destroyed when sub-program returns while execution contents of activation record are constantly changing as assignments are made to local variables

3) Current Instruction Pointer (CIP): It points to the current instruction.

4) Current Environment Pointer (CEP): It points to current activation records

Implementation of Sub-programs



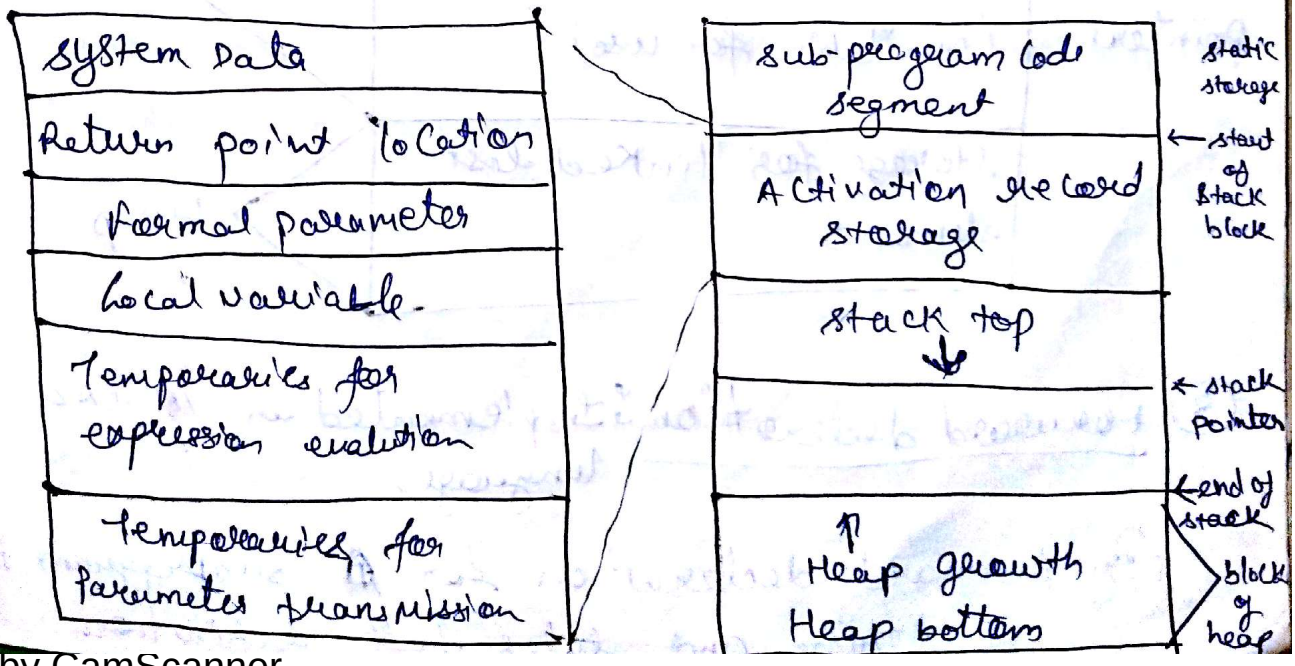
Procedure: An activation record for main program is created. In CEP a pointer is assigned. In CIP, first instruction of code is pointed. When a sub-program call is reached then activation record for sub-program is created with CEP & CIP. When return instruction is reached, it terminates activation of sub-program. Call instruction creates activation record, Stores old values of Instruction point (IP) and Environment point (EP) of CIP & CEP in

returned point and assign new IP, EP of CIP & CEP, thus control is transferred to called subprogram returned instruction fetches old IP, EP from return point & reinstates as values of CIP & CEP then control is returned to calling subprogram.

Subprogram implementation:

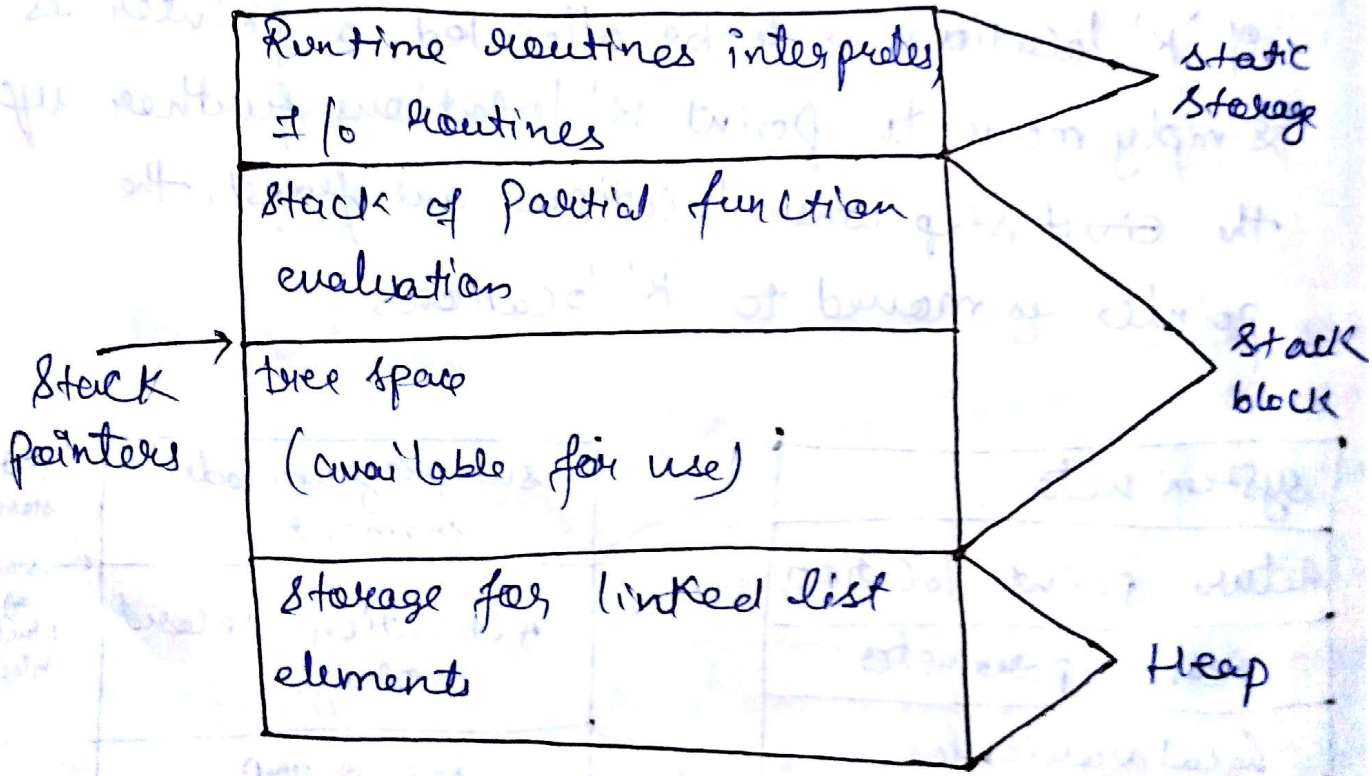
1) stack based
2) Forward declaration.
3) Recursion

1) Stack based: Implemented in C. Static pointer points to top of stack. All storage lies above the pointer. when a block of 'k' locations is to be allocated, a pointer is simply move to point 'k' locations further up the ~~civit~~ step. when locations are freed, the pointer is moved to 'k' locations.



2) Recursive Sub-programs: Each activation record contains a return point and temporaries temporaries for expression evaluation and parameter transmission.

Stack containing return point & temporaries may be hidden from programmer and allocated sequentially. LISP implementation also requires heap storage area which is managed to a free space list and garbage collection which is a special area and storage manager for full word data items such as numbers.



3) Forward declaration: Implemented in PASCAL language.

By forward declaration for a subprogram can be define and later full definition

of Body A is given -

forward declaration gives compiler an info. to correctly compile the call on A found in B, although the complete definition of A and not yet appeared.

```
procedure A
(formal parameter list); forward;
```

Scope of Subprogram :

Scope of subprogram is a region of the program where a defined variable can have its existance and beyond that variable can not be access. It defines which part of subprogram can access that variable.

Types of scope

→ Local scope
→ Global scope

Life-time of Subprogram :

It is the time period in which variable has value memory. Lifetime is defined when a variable is created & destroyed later.

Types of lifetime -

— Static lifetime
— Dynamic lifetime.

Static lifetime : A static variable is stored in data segment of object file of a program. Its lifetime is entire duration of

program execution. All compiler based languages like C, C++ are having static lifetime.

Dynamic lifetime: The lifetime begins when memory is allocated for object. All C interpreter based languages have dynamic lifetime eg LISP.

In C, malloc() & free() function exhibit dynamic lifetime.

Subprogram data control:

Accessibility of data at different points during program execution is termed as data control. It determines how data may be provided to each operation and how a result of one operation may be saved & retrieved.

Referencing Environment:

Each program or subprogram has a set of identifier association available for use during execution, this set of identifier is called referencing environment. Thus, it is a set of active binding. It is set up when sub-program activation is created and it remains unchanged during lifetime of the activation.

Types of referencing environment:

1) Local referencing environment
2) Nonlocal referencing environment
3) Global referencing environment.
4) Pre-defined referencing

1) Local referencing Environment : Set of associations created on entry to subprogram define only within that subprogram

2) Non-local referencing environment : Set up of associations for identifiers that may be use within a subprogram but are not created on entry to it. eg) structure.

3) Global referencing environment : If the association is created at the start of execution of main program are available to be use in the sub-program then it is global referencing environment.

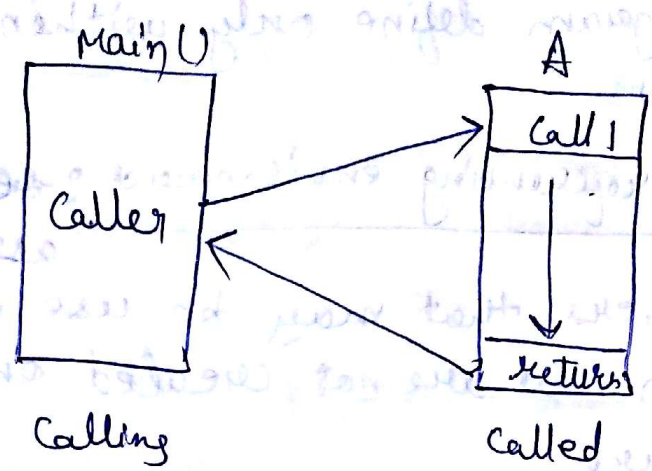
4) Pre-defined referencing environment : Some identifiers have a pre-defined association i.e., define directly in language definition. Any program may use it without explicitely creating it.

Subprogram : It is defined as self contained program which is written for the purpose of accomplishing some task.

Characteristics :

1) Each subprogram has a single entry point
2) Calling program is suspended during the execution. Called program gets executed. Only one sub-program is in execution at any given time.

3) control returns to calling sub-program when sub-program terminates.



Parts of Subprogram :

1) Subprogram header : It defines the program name, return type and ~~arg~~ argument declaration enclosed within pair of parenthesis.

2) Subprogram definition :

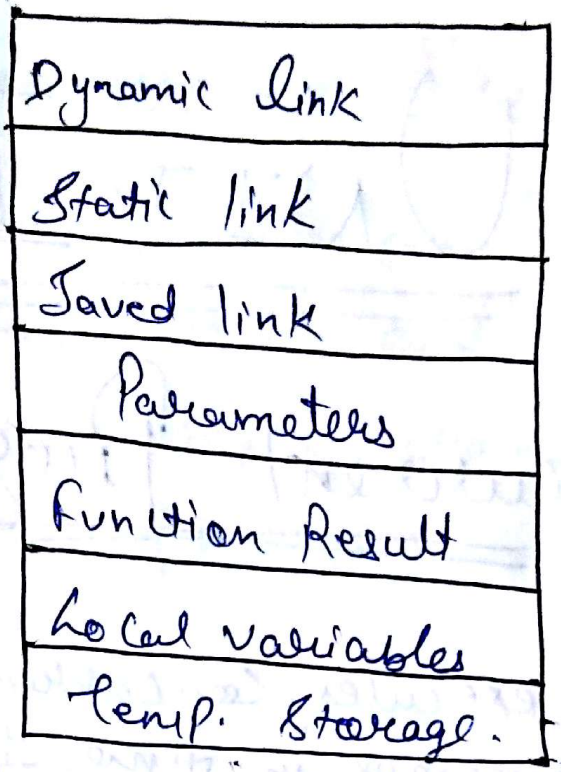
3) Subprogram call : use of function.

Advantages of Subprogram :

1) Minimization of memory space usage.
2) Simpling testing & debuging
3) Code reusibility.
4) Abstraction

Activation Record

It is data structure created each time a program is called and destroyed when subprogram returns.



i) Dynamic link : Points to that activation program is used in subprogram return.

ii) Static link : It points to activation used associated with nearest scope of the subprogram

iii) Saved link : It includes program counter & register counter at the time of subprogram are called saved link and is used for restoring the context of the caller of subprogram.

iv) Temporary storage : Used for evaluating expression, storage address