8.1 The Symbol Table

- **Definition**

> The symbol table is defined as the set of **Name** and **Value** pairs.

For example

	Name	Value
0	a	10
1	b	0
2	c	0
3		
4		
5		
6		

Fig. 8.1.1 Symbol table storing 3 identifiers

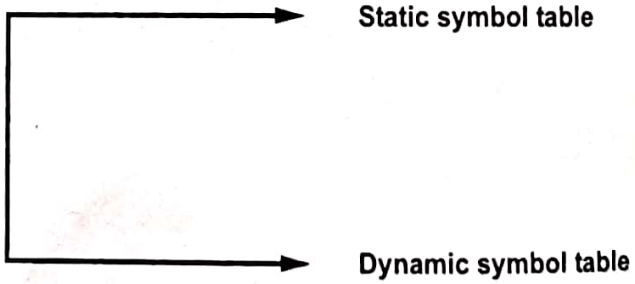
- **Use of Symbol Table**

The symbol tables are typically used in **compilers**. Basically compiler is a program which scans the application program (for instance : your C program) and produces machine code. During this scan compiler stores the identifiers of that application program in the symbol table. These identifiers are stored in the form of name, value address, type. Here the name represents the name of identifier, value represents the value stored in an identifier, the address represents memory location of that identifier and type represents the data type of identifier. Thus compiler can keep track of all the identifiers with all the necessary information.

- **Types of Symbol Tables**

There are two types of symbol tables.

The static symbol table stores the fixed amount of information whereas dynamic symbol table stores the dynamic information.



These symbol tables are normally used to implement static and dynamic data structures. Hence there can be static tree tables or dynamic tree tables, which are implemented using static and dynamic symbol tables respectively.

Examples of static symbol table : Optimal Binary Search Tree (OBST), Huffman's coding can be implemented using static symbol table.

Example of dynamic symbol table : An AVL tree is implemented using dynamic symbol table.

- **Advantages of using Symbol Tables**

Following are some advantages of using symbol tables. As symbol table is normally used in compiler, these advantages are relevant to compilers -

1. During compilation of source program, **fast look up** for the required identifiers is possible due to use of symbol table.
2. The runtime allocation for the identifiers is managed using symbol tables.
3. Use of symbol table allows to handle certain issues like scope of identifiers, and implicit declarations.

- **Operations on Symbol Table**

Following operations can be performed on symbol table -

1. Insertion of an item in the symbol table.
2. Deletion of any item from the symbol table.
3. Searching of desired item from symbol table.

8.2 Concept of Hashing

GTU : Summer-13 Marks 2

- **Hash Table** is a data structure used for storing and retrieving data very quickly. Insertion of data in the hash table is based on the key value. Hence every entry in the hash table is associated with some key. For example for storing an employee record in the hash table the employee ID will work as a key.
- Using the **hash key** the required piece of data can be searched in the hash table by few or more key comparisons. The searching time is then dependant upon the size of the hash table.
- The effective representation of dictionary can be done using hash table. We can place the dictionary entries (key and value pair) in the hash table using hash function.

Review Question

1. *Define Hashing.* **GTU : Summer-13, Marks 2**

8.3 Hashing Functions

GTU : Dec.-10, Winter-12, 13, Summer-14, Marks 7

- Hash function is a function which is used to put the data in the hash table. Hence one can use the same hash function to retrieve the data from the hash table. Thus hash function is used to implement the hash table.
- The integer returned by the hash function is called hash key.

For example : Consider that we want place some employee records in the hash table. The record of employee is placed with the help of key : employee ID. The employee ID is a 7 digit number for placing the record in the hash table. To place the record, the key 7 digit number is converted into 3 digits by taking only last three digits of the key.

If the key is 496700 it can be stored at 0^{th} position. The second key is 8421002, the record of this key is placed at 2^{nd} position in the array.

Hence the hash function will be –

H(key)=key%1000.

Where key%1000 is a hash function and key obtained by hash function is called hash key. The hash table will be –

	Employee ID	Record
0	4968000	
1		
2	7421002	
.	.	.
.	.	.
.	.	.
.	.	.
396	4618396	
397	4957397	
399	7886399	
.	.	.
.	.	.
.	.	.
998	7886998	
999	0001999	

- **Bucket and Home bucket :** The hash function H(key) is used to map several dictionary entries in the hash table. Each position of the hash table is called bucket.

  The function H(key) is home bucket for the dictionary with pair whose value is *key*.

Types of Hash function –

There are various types of hash functions that are used to place the record in the hash table –

1. **Division method :** The hash function depends upon the remainder of division. Typically the divisor is table length. For example :-

If the record 54, 72, 89, 37 is to be placed in the hash table and if the table size is 10

H(key) = record%table size

= 54 % 10

= 72 % 10

= 89 % 10

= 37 % 10

0	
1	
2	72
3	
4	54
5	
6	
7	37
8	
9	89

2. **Mid square :** In the mid square method, the key is squared and the middle or mid part of the result is used as the index.

If the key is a string, it has to be preprocessed to produce a number.

Consider that if we want to place a record 3111 then

$3111^2 = 9678321$

For the hash table of size 1000

H(3111) = 783 (the middle 3 digits)

3. **Multiplicative hash function :** The given record is multiplied by some constant value. The formula for computing the hash key is –

H(key)=floor(p *(fractional part of key*A)) where p is integer constant and A is constant real number.

Donald Knuth suggested to use constant A =0.61803398987

If key 107 and p=50 then

H(key) = floor(50*(107*0.61803398987)

= floor(3306.4818458045)

= 3306

At 3306 location in the hash table the record 107 will be placed.

4. **Digit folding :** The key is divided into separate parts and using some simple operation these parts are combined to produce the hash key.

For example, consider a record 12365412 then it is divided into separate parts as 123 654 12 and these are added together

$$H(key) = 123+654+12$$
$$= 789$$

The record will be placed at location 789 in the hash table.

5. **Digit analysis :** The digit analysis is used in a situation when all the identifiers are known in advance. We first transform the identifiers into numbers using some radix, r. Then we examine the digits of each identifier. Some digits having most skewed distributions are deleted. This deleting of digits is continued until the number of remaining digits is small enough to give an address in the range of the hash table. Then these digits are used to calculate the hash address.

Review Questions

1. *What do you mean by hashing ? What are various hash functions ? Explain each one in brief.* **GTU : Dec.-10, Marks 7**
2. *Explain different Hash function methods.* **GTU : Winter-12, Marks 7**
3. *What do you mean by Hashing ? Explain any FOUR hashing techniques.* **GTU : Winter-13, Summer-14, Marks 7**

8.4 Collision

Definition : *The situation in which the hash function returns the same hash key(home bucket) for more than one record is called collision and two same hash keys returned for different records is called synonym.*

Similarly when there is no room for a new pair in the hash table then such a situation is called overflow. Sometimes when we handle collision it may lead to overflow conditions. Collision and overflow show the poor hash functions.

For example :

Consider a hash function.

H(key) = recordkey%10 having the hash table of size 10.

The recordkeys to be placed are

131, 44, 43, 78, 19, 36, 57 and 77

0	
1	131
2	
3	43
4	44
5	
6	36
7	57
8	78
9	19

Now if we try to place 77 in the hash table then we get the hash key to be 7 and at ...n 7 already the recordkey 57 is place. This situation is called collision. From the ...n 7 if we look for next vacant passion at subsequent indices 8,9 then we find that ...e is no room to place 77 in the hash table. This situation is called overflow.

...racteristics of Good Hashing Function –

1. The hash function should be simple to compute.
2. Number of collisions should be less while placing the record in the hash table. Ideally no collision should occur. Such a function is called perfect hash function.
3. Hash functions should produce such a keys(buckets) which will get distributed uniformly over an array.
4. The hash function should depend upon every bit of the key. Thus the hash function that simply extracts the portion of a key is not suitable.

5 Collision Resolution Techniques GTU : May-11, 12, Summer-13, Marks 7

If collision occurs then it should be handled by applying some techniques. Such a ...nique is called **collision handling technique.**

There are two methods for detecting collisions and overflows in the hash table :

1. Chaining
2. Open addressing (linear probing)

Two more difficult collision handling techniques are –

1. Quadratic probing
2. Double hashing

8.5.1 Chaining

In collision handling method chaining is a concept which introduces an additional field with data i.e. chain. A separate chain table is maintained for colliding data. When collision occurs then a linked list(chain) is maintained at the home bucket.

For example :

Consider the keys to be placed in their home buckets are

131, 3, 4, 21, 61, 24, 7, 97, 8, 9

Then we will apply a hash function as

H(key) = key%D

where D is the size of table. The hash table will be -

Here D = 10.

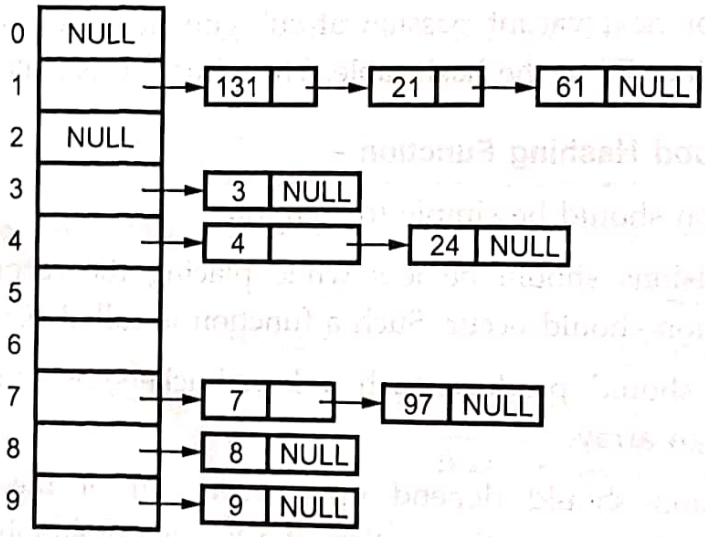


Fig. 8.5.1 Chaining

A chain is maintained for colliding elements. For instace 131 has a home bucket (key) 1. Similarly keys 21 and 61 demand for home bucket 1. Hence a chain is maintained at index 1. Similarly the chain at index 4 and 7 is maintained.

8.5.2 Open Addressing-Linear Probing

This is the easiest method of handling collision. When collision occurs i.e. when two records demand for the same home bucket in the hash table then collision can be solved by placing the second record linearly down whenever the empty bucket is found. When use linear probing(open addressing), the hash table is represented as a one-dimensional array with indices that range from 0 to the desired table size-1. Before inserting any elements into this table, we must initialize the table to represent the situation where all slots are empty. This allows us to detect overflows and collisions when we insert elements into the table. Then using some suitable hash function the element can be inserted into the hash table.

For example :

Consider that following keys are to be inserted in the hash table.

131, 4, 8, 7, 21, 5, 31, 61, 9, 29

Initially, we will put the following keys in the hash table.

131, 4, 8, 7.

We will use Division hash function. That means the keys are placed using the formula

$$H(key) = key\%tablesize$$

$$H(key) = key\%10$$

For instance the element 131 can be placed at

$$H(key) = 131\%10$$

$$= 1$$

Index 1 will be the home bucket for 131. Continuing in this fashion we will place 4, 8 and 7.

Index	Key
0	NULL
1	131
2	NULL
3	NULL
4	4
5	NULL
6	NULL
7	7
8	8
9	NULL

Now the next key to be inserted is 21. According to the hash function

$$H(key) = 21\%10$$

$$H(key) = 1.$$

But the index 1 location is already occupied by 131 i.e. collision occurs. To resolve this collision we will linearly move down and at the next empty location we will prob the element. Therefore 21 will be placed at the index 2. If the next element is 5 then we

get the home bucket for 5 as index 5 and this bucket is empty so we will put the element 5 at index 5.

Index	Key
0	NULL
1	131
2	21
3	NULL
4	4
5	5
6	NULL
7	7
8	NULL
9	NULL

After placing record keys 31, 61 the hash table will be

Index	Key
0	NULL
1	131
2	21
3	31
4	4
5	5
6	61
7	7
8	8
9	NULL

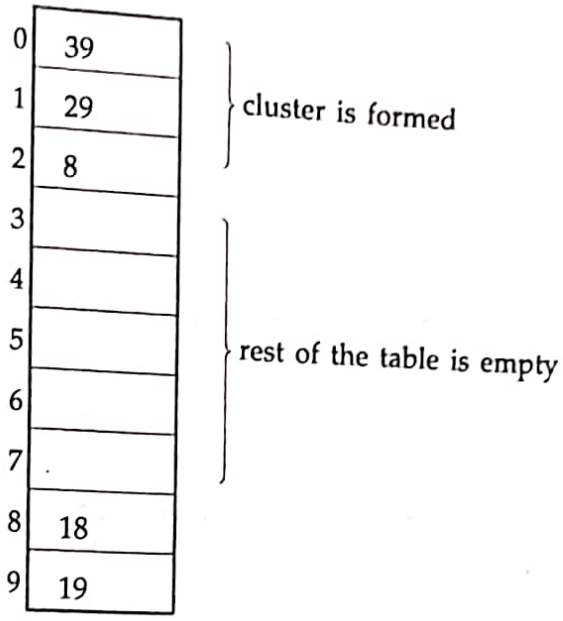
The next recordkey that comes is 9. According to decision hash function it demands for the home bucket 9. Hence we will place 9 at index 9. Now the next final recordkey is 29 and it hashes a key 9. But home bucket 9 is already occupied. And there is no next empty bucket as the table size is limited to index 9. The overflow occurs. To handle it we move back to bucket 0 and is the location over there is empty 29 will be placed at 0^{th} index.

Problem with linear probing

One major problem with linear probing is primary clustering. Primary clustering is a process in which a block of data is formed in the hash table when collision is resolved.

For example :

19 % 10 = 9
18 % 10 = 8
39 % 10 = 9
29 % 10 = 9
8 % 10 = 8



This clustering problem can be solved by quadratic probing.

```
/*****************************************************************
   Program to create hash table and handle the collision using linear probing. In this
Program hash function is (number %10)
*****************************************************************/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10
void main()
{

int a[MAX],num,key,i;
char ans;
int create(int);
void linear_prob(int [],int,int),display(int []);
clscr();
printf("\nCollision Handling By Linear Probing");
for(i=0;i<MAX;i++)
a[i]= -1;
do
{

printf("\n Enter The Number");
scanf("%d",&num);
key=create(num);/*returns hash key*/
linear_prob(a,key,num);/*collision handled by linear probing*/
printf("\n Do U Wish To Continue?(y/n)");
ans=getche();
}while(ans=='y');
display(a);/*displays hash table*/
```

```
...ter else*/
...end*/
... display(int a[MAX])

...
...("\n The Hash Table is...\n");
...=0;i<MAX;i++)
...("\n %d   %d",i,a[i]);
```

Output

```
...sion Handling By Linear Probing
...er The Number131
...U Wish To Continue?(y/n)y
...er The Number21
...U Wish To Continue?(y/n)y
...er The Number3
...U Wish To Continue?(y/n)y
...er The Number4
...U Wish To Continue?(y/n)y
...er The Number5
...U Wish To Continue?(y/n)y
...er The Number8
...U Wish To Continue?(y/n)y
...er The Number9
...U Wish To Continue?(y/n)y
...er The Number18
...U Wish To Continue?(y/n)n
...e Hash Table is...

 18
 131
 21
 3
 4
 5
 -1
 -1
 8
 9
```

Chaining without Replacement

...collision handling method chaining is a concept which introduces an additional ...d with data i.e. chain. A separate chain table is maintained for colliding data. When ...ision occurs we store the second colliding data by linear probing method. The

address of this colliding data can be stored with the first colliding element in the chain table, without replacement.

For example consider elements,

131, 3, 4, 21, 61, 6, 71, 8, 9

Index	Data	Chain
0	–1	–1
1	131	2
2	21	5
3	3	–1
4	4	–1
5	61	7
6	6	–1
7	71	–1
8	8	–1
9	9	–1

Fig. 8.5.2 Chaining without replacement

From the example, you can see that the chain is maintained the number who demands for location 1. First number 131 comes we will place at index 1. Next comes 21 but collision occurs so by linear probing we will place 21 at index 2, and chain is maintained by writing 2 in chain table at index 1 similarly next comes 61 by linear probing we can place 61 at index 5 and chain will be maintained at index 2. Thus any element which gives hash key as 1 will be stored by linear probing at empty location but a chain is maintained so that traversing the hash table will be efficient.

The drawback of this method is in finding the next empty location. We are least bothered about the fact that when the element which actually belonging to that empty location cannot obtain its location. This means logic of hash function gets disturbed. Let us now see a 'C' program which implements chaining without replacement.

```
/*****************************************************************
      Program to create hash table and handle the collision using chaining Without
replacement. In this Program hash function is (number%10)
*****************************************************************/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10
```

3	3	– 1
4	2	– 1
5	– 1	– 1
6	– 1	– 1
7	– 1	– 1
8	– 1	– 1
9	– 1	– 1

8.5.4 Chaining with Replacement

As previous method has a drawback of loosing the meaning of the hash function, to overcome this drawback the method known as changing with replacement is introduced. Let us discuss the example to understand the method. Suppose we have to store following elements :

131, 21, 31, 4, 5

0	– 1	– 1
1	131	2
2	21	3
3	31	– 1
4	4	– 1
5	5	– 1
6		
7		
8		
9		

Now next element is 2. As hash function will indicate hash key as 2 but already at index 2. We have stored element 21. But we also know that 21 is not of that position at which currently it is placed.

Hence we will replace 21 by 2 and accordingly chain table will be updated. See the table :

Index	data	chain
0	– 1	– 1
1	131	6
2	2	– 1
3	31	– 1

4	4	– 1
5	5	– 1
6	21	3
7	– 1	– 1
8	– 1	– 1
9	– 1	– 1

he value –1 in the hash table and chain table indicate the empty location.

he advantage of this method is that the meaning of hash function is preserved. But time some logic is needed to test the element, whether it is at its proper position.

```
**********************************************************
 Program to create hash table and handle the collision using chaining With
acement. In this Program hash function is (number%10)
**********************************************************/
clude<stdio.h>
clude<conio.h>
clude<stdlib.h>
fine MAX 10
[MAX][2];

main()

num,key,i;
r ans;
create(int);
d chain(int,int),display();

scr();
ntf("\nChaining With Replacement");
(i=0;i<MAX;i++)

][0]= -1;
][1]= -1;


rintf("\n Enter The Number");
canf("%d",&num);
ey=create(num); /*create hash key*/
hain(key,num);
rintf("\n Do U Wish To Continue?(y/n)");
ns=getche();
while(ans=='y');
isplay();
```

8.3.3 Quadratic Probing

Quadratic probing operates by taking the original hash value and adding successive values of an arbitrary quadratic polynomial to the starting value. This method uses following formula –

$$H_i(key) = (Hash(key)+i^2)\%m$$

where m can be a table size or any prime number.

For example : If we have to insert following elements in the hash table with table size 10 :

37, 90, 55, 22, 17, 49, 87.

We will fill the hash table step by step

37 % 10 = 7

90 % 10 = 0

55 % 10 = 5

22 % 10 = 2

11 % 10 = 1

0	90
1	11
2	22
3	
4	
5	55
6	
7	37
8	
9	

Now if we want to place 17 a collision will occur as 17 % 10 = 7 and bucket 7 has already an element 37. Hence we will apply quadratic probing to insert this record in the hash table.

$H_i(key) = (Hash(key) + i^2)\% m$

Consider i = 0 then

$(17+0^2) \% 10 = 7$

$(17+1^2) \% 10 = 8$, When i = 1

The bucket 8 is empty hence we will place the element at index 8.

Then comes 49 which will be placed at index 9.

49 % 10 = 9

where pointer 'Head' been caught in pointer 'T' and pointer head and 'P' are global variables

1. [Allocating memory] call getNode(T).

0	90
1	
2	22
3	
4	
5	55
6	
7	37
8	17
9	49

Fig. 8.5.3

if (DATA (T) <= 0) RIGHT (LEFT (T)) ← HEAD

LEFT (HEAD) ← (LEFT(T))

P ← LEFT(T)

Now to place 87 we will use quadratic probing.

DATA (T) ← 'y2'

LEFT (T) ← S

RIGHT (T) ← 0

RIGHT (S) ← T.

S ← T

2. Checking value

0	90
1	11
2	22
3	
4	
5	55
6	87
7	37
8	17
9	49

Fig. 8.5.4

3. call remove node (T)

return

else

call create -(D(T));

3. FINISH

return.

$(87 + 0)\%10 = 7$

$(87 + 1)\%10 = 8$ ... but already occupied

$(87 + 2^2)\%10 = 1$ ... already occupied

$(87 + 3^2)\%10 = 6$

It is observed that if we want to place all the necessary elements in the hash table the size of divisor (m) should be twice as large as total number of elements.

8.5 Double Hashing

Double hashing is technique in which a second hash function is applied to the key when a collision occurs. By applying the second hash function we will get the number of positions from the point of collision to insert.

There are two important rules to be followed for the second function :

- It must never evaluate to zero.
- Must make sure that all cells can be probed.

The formula to be used for double hashing is

$$H_1(key) = key \bmod tablesize$$

$$H_2(key) = M - (key \bmod M)$$

where M is a prime number smaller than the size of the table.

Consider the following elements to be placed in the hash table of size 10.

37, 90, 45, 22, 17, 49, 55

Initially insert the elements using the formula for $H_1(key)$.

Insert 37, 90, 45, 22.

$H_1(37) = 37 \% 10 = 7$

$H_1(90) = 90 \% 10 = 0$

$H_1(45) = 45 \% 10 = 5$

$H_1(22) = 22 \% 10 = 2$

$H_1(49) = 49 \% 10 = 9$

0	90
1	
2	22
3	
4	
5	45
6	
7	37
8	
9	49

Now if 17 is to be inserted then

$H_1(17) = 37 \% 10 = 7$

$H_2(key) = M - (key \% M)$

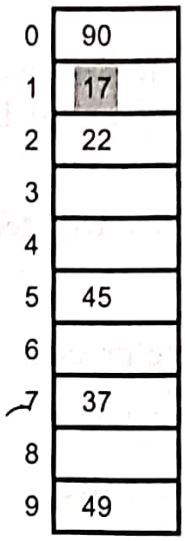


Fig. 8.5.5

Here M is a prime number smaller than the size of the table. Prime number smaller than table size 10 is 7.

Hence M = 7

$H_2(17) = 7 - (17\%7) = 7 - 3 = 4$

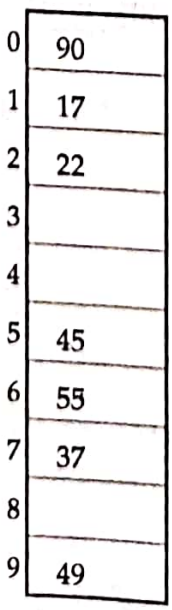
That means we have to insert the element 17 at 4 places from 37. In short we have to take 4 jumps. Therefore the 17 will be placed at index 1.

Now to insert number 55.

$H_1(55) = 55\%10=5$ ... collision

$H_2(55) = 7 - (55\%7) = 7 - 6 = 1$

That means we have to take one jump from index 5 to place 55. Finally the hash table will be –



parison of quadratic probing and double hashing

double hashing requires another hash function whose probing efficiency is same another hash function required when handling random collision.

double hashing is more complex to implement than quadratic probing. The dratic probing is fast technique than double hashing.

Rehashing

rehashing is a technique in which the table is resized, i.e., the size of table is doubled creating a new table. It is preferable if the total size of table is a prime number. re are situations in which the rehashing is required-

When table is completely full.

With quadratic probing when the table is filled half.

When insertions fail due to overflow.

such situations, we have to transfer entries from old table to the new table by mputing their positions using suitable hash functions.

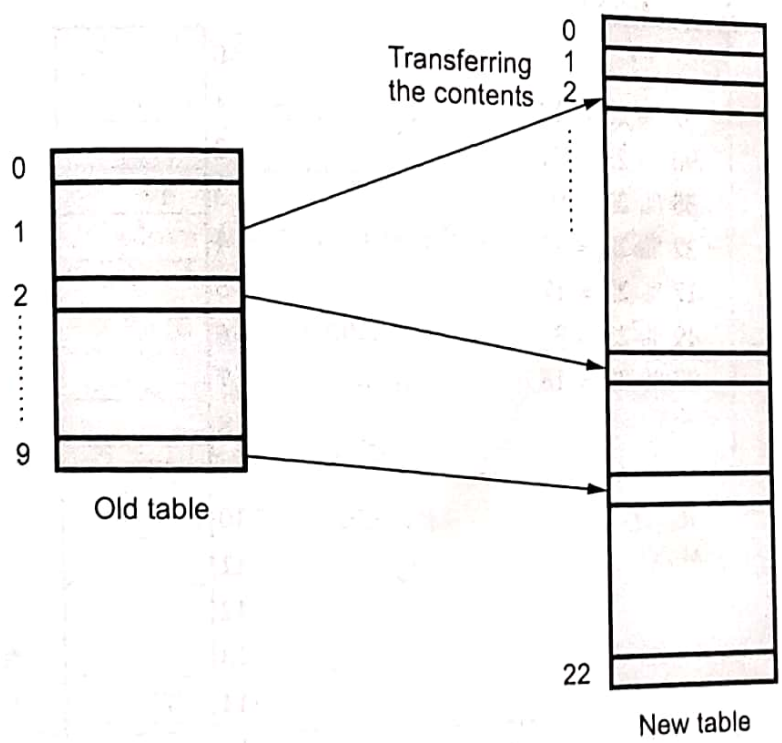


Fig. 8.5.6 Rehashing

Consider we have to insert the elements 37, 90, 55, 22, 17, 49 and 87. The table size is and will use hash function,

H(key) = key mod tablesize

37 % 10 = 7
90 % 10 = 0
55 % 10 = 5
22 % 10 = 2
17 % 10 = 7 Collision solved by linear probing, by placing it at 8
49 % 10 = 9

0	90
1	
2	22
3	
4	
5	55
6	
7	37
8	17
9	49

Now this table is almost full and if we try to insert more elements collisions will occur and eventually further insertions will fail. Hence we will rehash by doubling the table size. The old table size is 10 then we should double this size for new table, that becomes 20. But 20 is not a prime number, we will prefer to make the table size as 23. And new hash function will be

H(key) = key mod 23

37 % 23 = 14
90 % 23 = 21
55 % 23 = 9
22 % 23 = 22
17 % 23 = 17
49 % 23 = 3
87 % 23 = 18

0	
1	
2	
3	49
4	
5	
6	
7	
8	
9	55
10	
11	
12	
13	
14	37
15	
16	
17	17
18	87
19	
20	
21	90
22	22

...the hash table is sufficiently large to accommodate new insertions.

...ntages

1. This technique provides the programmer a flexibility to enlarge the table size if required.
2. Only the space gets doubled with simple hash function which avoids occurrence of collisions.

...ple 8.5.1 *The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function h(k) = k mod 10 and linear probing. What is the resultant hash table ?* **GTU : Summer-13, Marks 7**

...tion : Given data :

The hash function is h(k) = k mod 10

Hash table length = 10

12 % 10 = 2

18 % 10 = 8

13 % 10 = 3

2 % 10 = 2

Collision occurs

∴ By linear probing

Move down and place 2 at index 4.

3 % 10 = 3 ← Collision

23 % 10 = 3 ← Collision

5 % 10 = 5 ← Collision

15 % 10 = 5 ← Collision

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

Hash Table

...view Questions

1. *What is hashing ?Explain hash clash and its resolving techniques* **GTU : May-11, Marks 7**
2. *Define hash clash. Explain primary clustering, secondary clustering, rehashing, and double hashing.* **GTU : May-11, Marks 7**
3. *Explain the basic two techniques for Collision-resolution in Hashing with example. Also explain primary clustering.* **GTU : May-12, Marks 5**

9.1 Concepts of Fields, Records and File

A **file** is a collection of records. **Record** is nothing but the source of information. It typically stored in rows in the file structure. **Field** is a piece of data which is associated with the entire record. It is refered as column.

For example : Assume that we have a file "Student.dat" having the contents as follows -

Roll No.	Name	Marks
1	John	66
2	Mathew	70
3	Steve	60

So by observing above file, we can say that this file is a collection of 3 records. These records can be written in File with the associated fields such as Roll_No, Name and Marks. To identify each record, a unique key is associated with it. In above file **Roll_No** can be thought of as a **key**. Using a key we can access information from file. We must choose unique field as a key. This key is called the primary key of file.

9.2 Operations on File

1. Declaration of file Pointer :

Syntax: - FILE *filepointer;

Example : FILE *fp;

FILE is a structure which is defined in "stdio.h". Each file we open will have its own FILE structure. The structure contains information like usage of the file, its size, memory location and so on. The fp is a pointer variable which contains the address of the structure FILE.

2. Creating a file / Opening a file :

Syntax : filepointer = fopen("filename","mode");

where filename is the name of file you want to create or open

mode can be – read - "r"

write - "w"

append - "a"

Depending on the type of file we want to open/create we use –"rb", "wb", or just "r", "w", "a".

Using "r" opens the file in read mode. If the file does not exist, it will create a new file, but such a file created is of no use as you cannot write to any file in read mode.

Using "r+" allows to read from file as well as write to the file.

Using "w" opens a file in write mode. If the file does not exist, it will create a new file.

Using "w+" allows you to write contents to file, read them as well as modify existing contents.

Using "a" opens the file in append mode. If the file does not exist, then it creates a new file. Opening a file in append mode allows you to append new contents at the end of the file.

Using "a+" allows to append a file, read existing records, cannot modify existing records.

Example :

```
fp = fopen("Student.dat","w");
```

3. Closing the file :

Syntax : fclose(filepointer);

Example :

```
fclose(fp);  /* closes the file pointed by the fp */
```

4. Setting the pointer to start of file :

Syntax : rewind(filepointer);

Example :

```
rewind(fp);  /* places pointer at the start of the file */
```

5. Writing a character to file :

Syntax: fputc(character, filepointer);

Example :

Suppose we have

```
char ch='a';
```

To write the character to the file, we use

```
fputc(ch,fp);
```

Note If the file is opened in "w" mode, it will overwrite the contents of the entire file and write the only character we want to write. To write it at the end of the file, open the file in "a+" mode.

6. Reading a character from file :

Syntax : fgetc(filepointer);

Example :

```
char ch;  /* ch is the character where you are going to get the char
  from file*/
        ch = fgetc(fp);
```

7. Writing string to file:

Syntax : fputs(string, filepointer);

Example :

Suppose you have a string

```
char s[] ="abcd";
```

then we write as

```
fputs(s,fp);
```

The above operation writes the string "abcd" to the file pointed by fp.

8. Reading string from file:

Syntax: fgets(stringaddress, length, filepointer);

Example :

```
char s[80];  /* declared a string into which we want  to read the
                string from file */
fgets(s,79,fp);  /* Will read the 78, (79–1) characters of string from
                file into s */
```

9. Writing of characters, strings, integers,floats to file :

Syntax:

```
fprintf(filepointer, "format string", list of variables);
```

We are already aware of the printf function. fprintf has the same arguments as that of printf, addition to which filepointer is added. The variables included in the fprintf are written to the file. But how can we write the data without knowing their values, so we need to have a scanf function before it and then can write the data to file. You will understand this better by following example.

'C' Program

```
/*****************************************************
   Program for introducing the file premitives such as
   fopen,fclose,fprintf.
*****************************************************/
#include<stdio.h>
/*
The main Function
Input:none
Output:none
Called By:O.S.
*/
main()
{
FILE *fp;
int rno;
char name[20];
clrscr();

fp = fopen("Student.dat","w");
if(fp==NULL)
{
printf("File opening error");
exit(1);
}
printf("Enter the rollno and name:");
scanf("%d %s",&rno,&name);      /* asking user to enter data */
fprintf(fp,"%d %s",&rno,&name);  /* writing data to file */

fclose(fp);
getch();
return;
}
/**********************End Of Program**********************/
```

Output

```
Enter the rollno and name:
1
aa
```

student.dat

```
1aaa
```

10. Reading of variables from file :

Syntax:

fscanf(filepointer,"format string",list of addresses of variables);

Having similar arguments as that of the fprintf statement, the fscanf reads the contents of the file into the variables specified.

Example :

```
fscanf(fp,"%d  %s",&rno,&name);
printf("Read data is: %d  %s",rno,name);
```

Note *The fprintf and fscanf statements are useful only when the data we want to enter is less, or rather they have disadvantage that when the number of variables increase, writing them or reading them becomes clumsy.*

11. Writing contents to file through structure :

Syntax : fwrite(pointer to struct,sizeof struct,no.of data items,filepointer);

'C' Program

```
/*************************************************************
Program for file primitives such as fopen,fwrite.This program writes the string to the file.

*************************************************************/
#include<stdio.h>
#define MAX 20
struct stud  /* structure declaration */
{
 int rno;
 char name[MAX];
};
/*
The main Function
Input :none
Output:none
*/
main()
{
struct stud s;
FILE *fp;
clrscr();
s.rno=1;                /* fill the structure contents*/
strcpy(s.name,"Sachin");
fp = fopen("Student.dat","w");
if(fp==NULL)
{
printf("File opening error");
exit(1);
}

fwrite(&s,sizeof(s),1,fp);  /* write the structure contents to file*/
fclose(fp);
```

```
return;
}
/**********End of Program**********
                    Output –student.dat
Sachin 0 ≅ + ÛI□2
```

Thus we can see from the output of the program that fwrite writes the data in the file in binary mode.

12. Reading contents of file through structure :

Syntax : fread(pointer to struct,sizeof struct, no.of data items, filepointer);

Example :

To read the contents written in file we can have:-

```
fread(&s,sizeof(s),1,fp);
```

Note *Prior to this statement it is necessary that the pointer is located at the proper position from where we want to read the file. Here you can use the rewind(filepointer) function.*

An fread or fwrite function moves the pointer to the beginning of the next record in that file.

13. Moving the pointer from one record to another :

Syntax: fseek(filepointer, offset, whence);

where

offset is difference in bytes between offset and whence position.

whence can be:

1) SEEK_SET move the pointer with reference to beginning of the file.
2) SEEK_CUR move the pointer with reference to its current position in file.
3) SEEK_SET move the pointer from end of file.

Example :

Let us take the example where we have written a record in file. Suppose we want to read that record, we have two options :

1) To use rewind function. 2) To use fseek.

We can use fseek in following fashion:

prior to the fclose(fp), we can have

```
fseek(fp,-sizeof(s),SEEK_CUR);
```

This will place the pointer to the beginning of the record we have currently written.Then we can make use of fread(...) to read that record again.

9.3 Types of Files

There are three types of file organizations -

1. Sequential file
2. Indexed file
3. Random file

Let us discuss them in detail.

9.4 Sequential Files

To understand the sequential files, lets start with an example. Consider the example of a tape or rather a cassette where the songs are stored sequentially. Whenever we want to play any song from it, we read it sequentially. Storing data sequentially is the simplest form of file, but a tedious one. Reading data to a file or writing data from it takes a lot of time as the data is not sorted. The data is stored on FCFS basis. Taking example where we want to retrieve a record which is unfortunately stored at the last position in the file requires to search the entire file. Hence the time required is very large in this case.

'C' Program

```
/*****************************************************************
Implementation of various file operations such as create, display, search and modification
on student database with USN,Name and marks of three subjects
*****************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
struct record
{
 int USN;
 char name[20];
 int marks1,marks2,marks3;
};
struct record r;
FILE *fp;
void main()
{
 int n,choice;
 char ch;
 void Create_file(int);
```

```
void Display_file(int);
void Modify_file();
struct record *Search_file();
clrscr();
printf("\n How many Records are there in the file?");
scanf("%d",&n);

do
{
clrscr();
printf("\n\t Main Menu");
printf("\n1. Create a file");
printf("\n2. Display a file");
printf("\n3. Search a file");
printf("\n4. Modify a file");
printf("\n5. Exit");
printf("\n Enter Your Choice ");
scanf("%d",&choice);
switch(choice)
{
  case 1:Create_file(n);
         break;
  case 2:Display_file(n);
         break;
  case 3:r=*Search_file();
         printf("\n USN       Name      marks1  marks2   marks3\n");
         printf("\n-----------------------------------------\n");
         flushall();
  printf(" %d     %s          %d      %d     %d \n",r.USN,r.name,r.marks1, r.marks2,r.marks3);
         printf("\n-----------------------------------------\n");
     break;
  case 4:Modify_file();
         break;
  case 5:exit(0);
}
printf("\n Do You want To Continue?");
ch=getch();
while(ch=='y'||ch=='Y');
etch();
}

oid Create_file(int n)
{
  int i;
  fp=fopen("stud.dat","a+");
  for (i=0;i<n; i++)
  {
       printf("\n Enter The name of the student ");
       flushall();
```

```
5. Exit
Enter Your Choice 2
Reading all the records sequentially
USN     Name      marks1    marks2    marks3
-------------------------------------------
1       aaa  40   50        60
2       bbb  55   66        77
50      XYZ  55   66        77
4       ddd  45   65        76
5       eee  32   43        40
-------------------------------------------
Do You want To Continue?
    Main Menu
    1. Create a file
    2. Display a file
    3. Search a file
    4. Modify a file
    5. Exit
    Enter Your Choice 5
```

Advantage of sequential file:

Sequential files are very simple to manage.

Disadvantages of sequential file:

1. Time required to retrieve any record is more as the entire file is searched.
2. Efficiency is less.

Owing to these drawbacks let us start with other file which will overcome these drawbacks.

Review Questions

1. *Explain the sequential file organization.*
2. *Write a C program for implementing the sequential file organization.*

9.5 Indexed Sequential File

GTU : May-12, Winter-12, Summer-13, 14, Marks 7

Since we are clear with sequential files, lets now got to other type which is 'Indexed Sequential' file format. There are many advantages of using an 'indexed sequential' over a normal sequential. Firstly, in an indexed sequential format, we maintain two files - 1. Normal sequential file and 2. An sorted index file.

What ever is our data we store it in the sequential file and in the index file we have the id or say the primary key of the sequential file along with the offset of that particular record in the sequential file.

To be more precise lets take the following example:

We have to maintain grossery details, where in the sequential file we have

itemno. name price type etc.

To refer to any item if we were using normal sequential file, we would have to scan the entire file to find that particular item details. But with the indexed sequential we have another index file which will in this case contain

itemno. offset

Another example is of students database the main file will have Roll, Name, Age, Marks, and the index file will have Roll offset.

This will be in sorted format for the index file. Hence whenever we are referring to any record, first the index file will be searched. From that search the offset is retrieved and the required record can be seek from the sequential file. See Fig. 9.5.1.

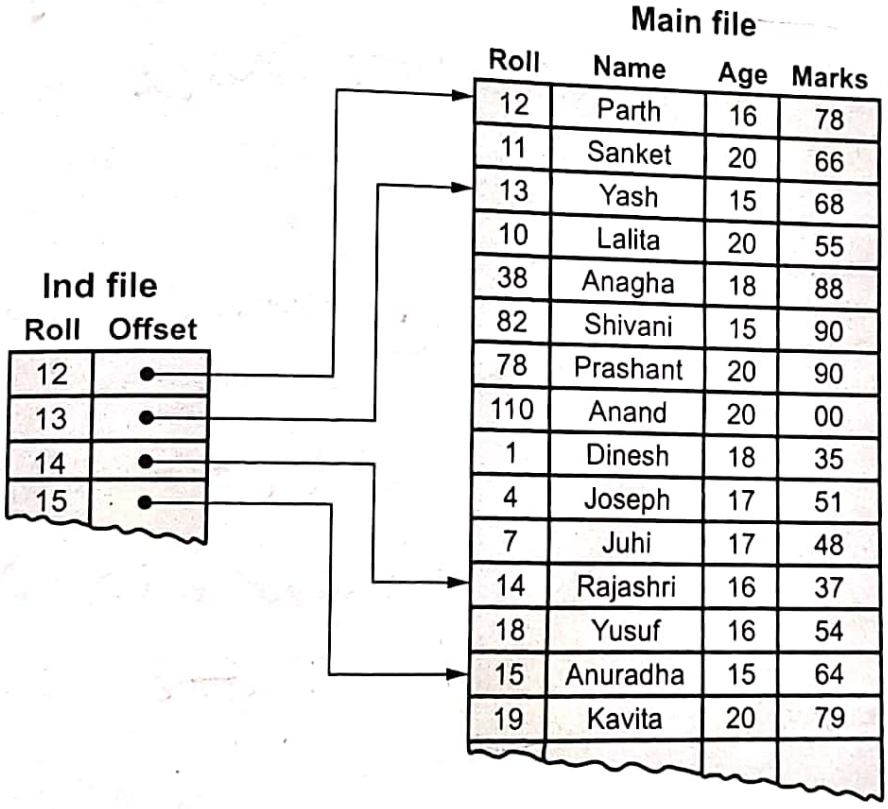


Fig. 9.5.1 Index sequential File

You will be more clear from the following sample figure and sample program.

Here we are storing various functions in function.cpp and structure in includes.h

C++ Program

```
/**************************************************************
   Program for the indexed sequential file.We have assumed the student database for
this type of file organization.
**************************************************************/
```

Review Question

1. *Write a short note on - Indexed sequential file.*

GTU : May-12, Summer-13, 14, Marks 4, Winter-12, Marks 7

9.6 Random File

Random organization is a kind of file organization in which records are stored at random locations on the disks.

There are three techniques used in random organization and those are given in following Fig. 9.6.1.

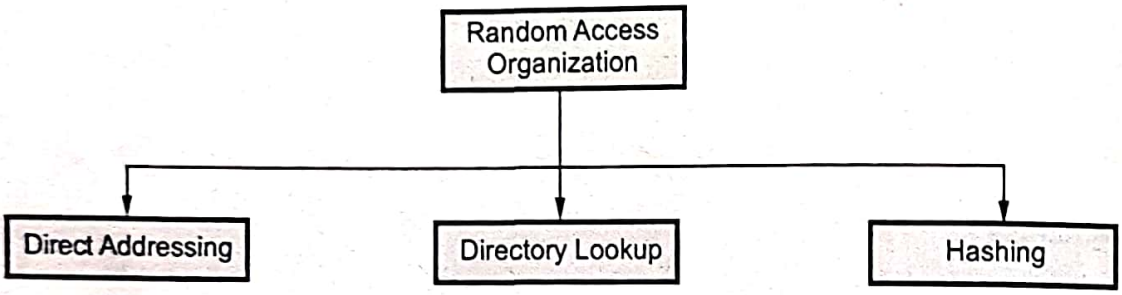


Fig. 9.6.1 Random access organization

Let us discuss each of them -

1. Direct Addressing

- In direct addressing two types of records are handled: fixed length record and variable length record.
- For storing the fixed length records the disk space is divided into the nodes. These nodes are large enough to hold individual record.
- Every fixed length record is stored in node number # which is equal to the primary key value. For example: If a primary key value is 185 then the record must be present in the node number 185.

For example

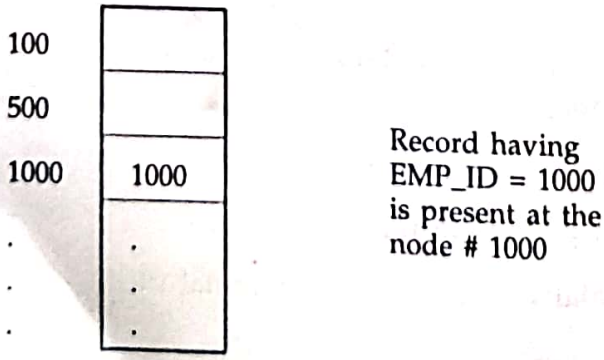


Fig. 9.6.2 Storing of fixed length record

- If we consider that the records are stored on the external storage devices then **deletion** and **searching** of the record requires one disk access. If we want to **update** a record then it requires two disk access, one for reading the record and one for writing the updated data back to the disk.
- For storing the **variable length records** on the disk, the address (pointer) of each individual record is stored in the file at specific index. We can locate the variable length record using the index of the pointer. This pointer will point to desired record which is present on the disk.

Variable length records make the storage management more **complex.**

Directory Lookup

- In this scheme the index for the pointers to the records is maintained.
- For retrieving the desired record first of all the index for the record address is searched and then using this record address the actual record is accessed.

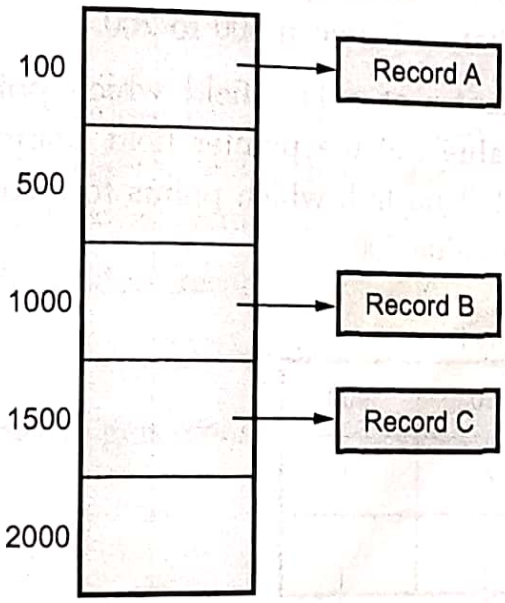


Fig. 9.6.3 Directory lookup

- The **drawback** of this method is that it requires more disk access than direct address method.
- **Advantage** of this method is that effective disk space utilization in it as compared to direct addressing method.

Hashing

- Hashing is a technique in which hash key is obtained using some suitable **hash function** and record is placed in the hash table with the help of this **hash key**.
- Thus in this random organization, the record can be quickly searched with the help of hash function being used.

- For creation of hash table the available file space is divided into **buckets** and **slots**.
- Some file space is left aside for handling the overflow situation.
- The total number of slots per bucket is equal to the total number of records each bucket can hold.

9.7 Multi-Key File Organization

GTU : Winter-13

For understanding multiple key access file organization consider the Roll_no record. In the Roll_no record structure, there are 3 fields - **value, length** and **pointer to the first record**. The value field indicates the upper bound value for the Roll_no. For instance : If the roll number of particular student is 437 then it lies between 0 to 500. Note that here upper bound is 500. Hence the record of that student must be associated with value 500. Similarly if roll_number of particular student is 689 then it must be associated with the value 700. The length field denotes total number of records. From Fig. 9.7.1 length 2 in the value of 500 means there are 2 records whose Roll_no lie between 0 to 500. Similarly length 2 for value 700 means there are 2 records who have Roll_no that lie between 500 to 700.

The third field is the pointer or a link field which points to the first record. For instance in Fig. 9.7.1. For **value 500** the pointer field points to record BBB and in the record BBB there is a field Roll_no link which points to record EEE. Thus there are total 2 records BBB and EEE with value 500.

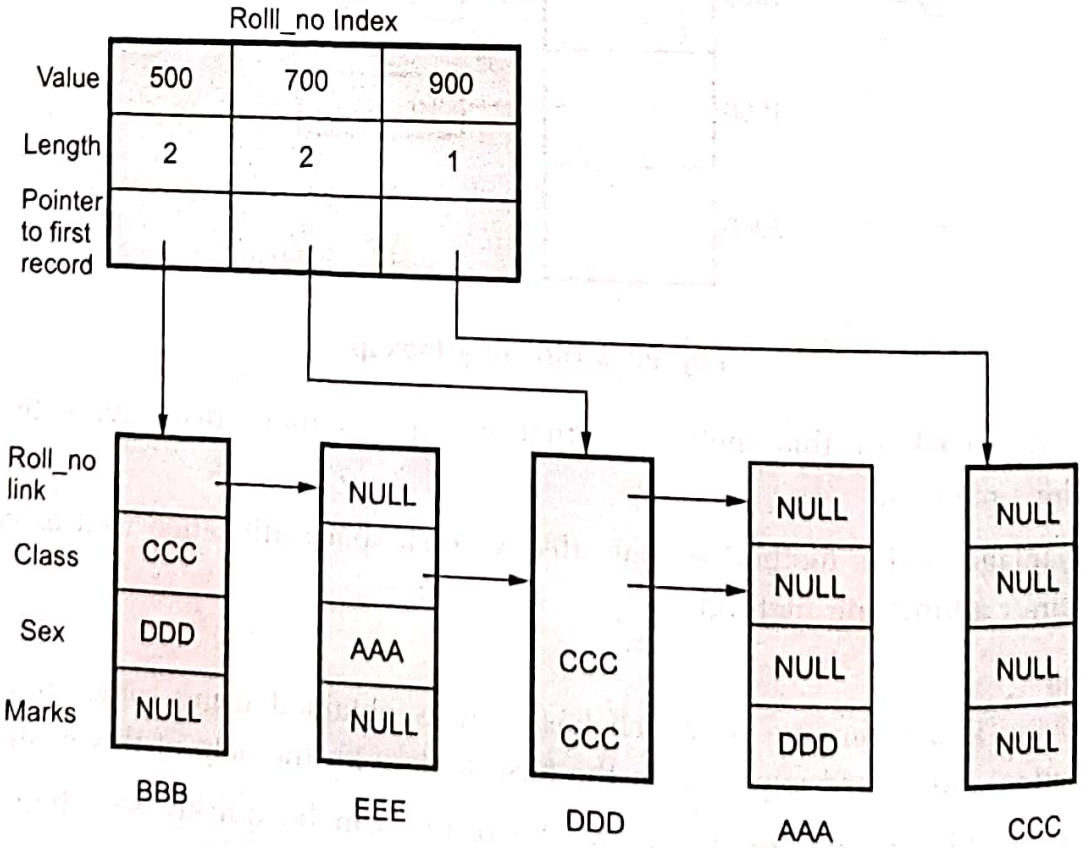


Fig. 9.7.1 Multi-list structure

Similarly for **value 700** there are 2 records the pointer field points to the first record DDD. The record DDD shows the next record of it by pointing to AAA (Refer the Roll_no link of record DDD).

And for **value 900** there is only one record i.e. CCC which is pointed by pointer field. The index for each key field is maintained which is useful for executing any query. Observe Fig. 9.7.1 of class index. This figure tells us that there are two records for fifth standard and 3 records for tenth standard. The first student of fifth standard class is BBB and second student is CCC. (Just refer the class field of record BBB from Fig. 9.7.1)

Thus we can solve the query "select * from stud_table where class = fifth" and the answer will be BBB and CCC. If we observe Fig. 9.7.1 of Marks index, the column of **second class** value shows that there are 3 records, out of which first record is AAA. Now from Fig. 9.7.1 record AAA has a Marks field which denotes next record as DDD and Marks field of record DDD denotes CCC as the next record. And for record CCC the Marks field denotes the value NULL. This all indicates the second class holder students are AAA, CCC and DDD.

Advantages

1) The multi-list structure provides satisfactory solution to simple and range queries. For instance : "Select * from dept_table where salary > 10000"
   Such queries can be executed efficiently using multi-list structure.
2) Quick access to every individual record is possible.

Disadvantage

1) Some amount of memory gets consumed in maintaining the link or address field.

Review Question

1. *Explain various multiple key access file organization in brief with advantages and disadvantages of each method.* **GTU : Winter-13, Marks 7**

9.8 Access Methods

GTU : Winter-12

The method by which records can be retrieved or updated from the file is called access method. Records within the file can be organized in variety of ways. Fig. 9.8.1 shows various types by which a file can be organized -

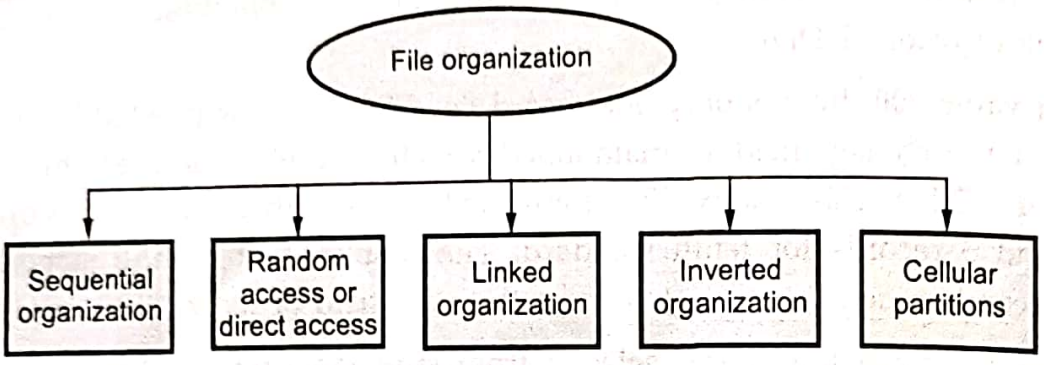
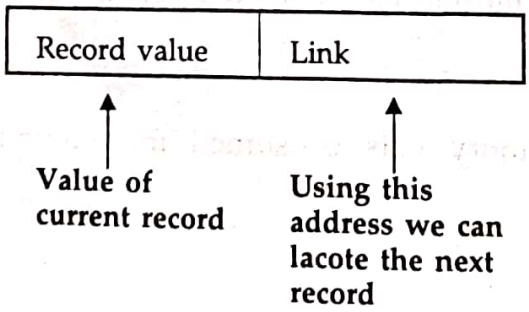


Fig. 9.8.1

9.8.1 Linked Organization

- In linked organization the logical sequence of the records is different than the physical sequence. In any sequential organization if we are accessing n^{th} node at Loc_i then $(n+1)^{th}$ record may be located at (Loc_i+c) where c is the constant which represents the length of the record or it may be some inter-record spacing.
- In linked organization we can access next logical record by following the **link-value** pair. The link-value pair denotes each individual record.
- The typical structure of every record is as follows -



Thus records in the linked organization can be stored as follows -

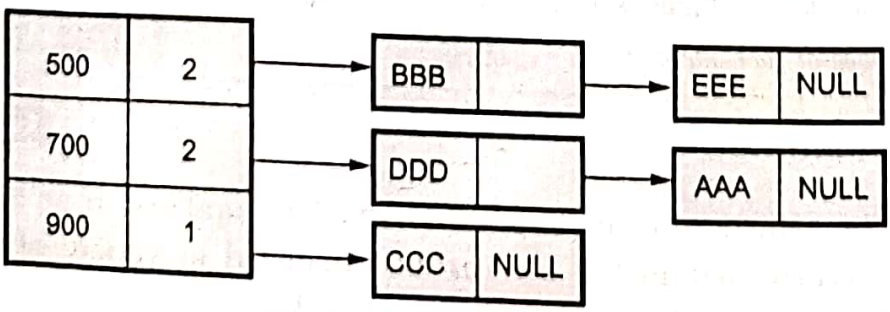


Fig. 9.8.2 Roll_No Index

9.8.2 Inverted File Organization

- Inverted files are similar to multi-lists.

- The difference between multi-list and inverted files is that in multi-lists records with the same key value are linked together along with the **link information** being kept in **individual record**. But in case of inverted files this **link information** is kept in the **index itself**.

For example

437	BBB
488	EEE
689	DDD
695	AAA
778	CCC

Fig. 9.8.3 (a) Roll_No index

Fifth	BBB, CCC
Tenth	EEE, DDD, AAA

Fig. 9.8.3 (b) Class index

Female	BBB, DDD, CCC
Male	EEE, AAA

Fig. 9.8.3 (c) Sex index

First class	EEE
Second class	AAA, DDD, CCC
Pass class	BBB

Fig. 9.8.3 (d) Marks index

Consider Fig. 9.8.3 (a) of Roll-no index which shows records BBB, EEE, DDD, AAA and CCC. In Fig. 9.8.3 (b) of class index two class are there fifth and tenth and we can observe that in the link information is stored in the index itself. Hence for fifth class records are BBB and CCC. And for tenth class records are EEE, DDD and AAA .

Similarly from sex index Fig. 9.8.3 (c), it is clear that BBB, DDD and CCC are females and EEE and AAA are males.

- The above index structure is a dense index structure **Dense Indexing**. The dense index is a kind of indexing in which record appers for every search key value in the file.

For example

- Thus in inverted files the index entries is of variable length. Hence inverted files structure is **more complex** than multi-list file structure.
- Following are the two steps that are adopted while searching a record from inverted files -

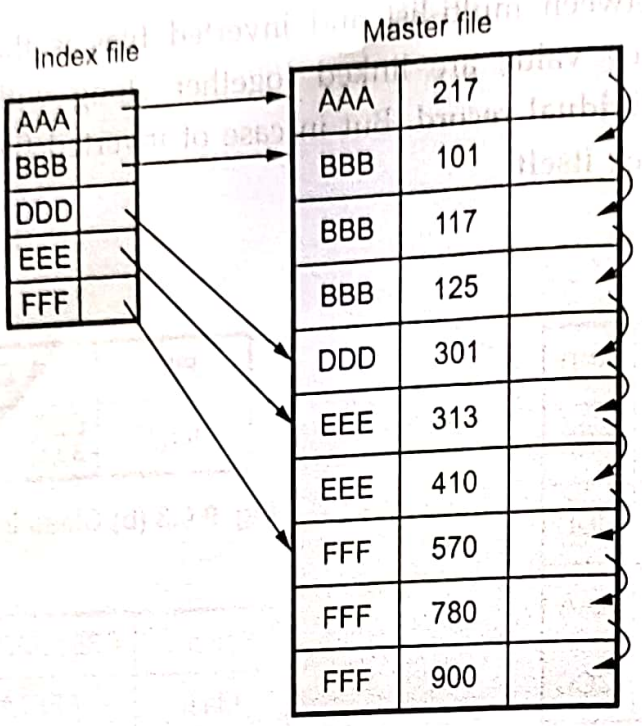


Fig. 9.8.4 Dense indexing

i) Index of required record is searched first of all.

ii) Then actual record is retrieved.

- In inverted files the index structure is important. The records can be arranged sequentially, randomly or linked depending on primary key.
- The number of disk accesses required = Number of records being retrieved + Processing for indexes.

Advantages

1) Inverted files are space saving as compared to other file structures when record retrieval does not require retrieval does not require retrieval of key fields.

Disadvantages

1) Insertion and deletion of records is complex because it requires the ability to insert and delete within indexes.

2) Index maintainance is complicated as compared to multi-list.

9.8.3 Cellular Partitions

- For reducing the searching time during file operations, the storage media (e.g. secondary memory, magnetic disk, magnetic tape etc.) may be divided into cells.
- The cells can be of two types -

i) Entire disk pack can be a cell

ii) A cylinder can be a cell.

- A list of records can occupy either entire disk pack or it may lie on particular cylinder.
- If all the records lie on the same cylinder then without moving the read/write head the records can be accessed.
- If the cell is nothing but entire disk pack then the disk is partitioned into different partitions. Such partitions are called **cellular partitions**. Then these different cells can be searched in parallel.

Advantages of cellular partitions

1) Various read operations can be performed parallely in order to reduce the search time.

2) Faster execution of any query.

Disadvantage

1) If multiple records lie in the same cell then reading a single cell becomes a time consuming process.

Review Questions

1. *Write a short note on inverted key file organization.* **GTU : Winter-12, Marks 7**
2. *Write a short note on Cellular Partitions.*

❑❑❑