Problem Statement

Topic: File Allocation Strategies

Objective: Write a program to simulate the following file allocation strategies.

a) Sequential b) Indexed c) Linked.

Problem Description

A file is a collection of data, usually stored on disk. As a logical entity, a file enables to divide data into groups. As a physical entity, a file should be considered in terms of its organization. The term "file organization" refers to the way in which data is stored in a file and, consequently, the method(s) by which it can be accessed.

a) Sequential File Allocation

In this file organization, the records of the file are stored one after another both physically and logically. That is, record with sequence number 16 is located just after the 15th record. A record of a sequential file can only be accessed by reading all the previous records.

B) Linked File Allocation

With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block.

C) Linked File Allocation

Indexed file allocation strategy brings all the pointers together into one location: an index block. Each file has its own index block, which is an array of disk-block addresses. The it entry in the index block points to the it block of the file. The directory contains the address of the index block. To find and read the it block, the pointer in the it index-block entry is used.

Solution Code

```
#include <iostream>

#include <windows.h>

#include <stdio.h>

using namespace std;
```

```
typedef struct file
{
   char fN[10];
   int sB, nB;

}fileTable;

typedef struct block
{
   int bno;
   struct block *next;

}Block;

typedef struct Node
{
   char name[10];
   int nob;
   Block *sb;

}nodeTable;

typedef struct stru
{
   char name[10];
   int numB;
   int Bnum[1000];
}Indexed;

void Sequential(); //Sequential File Allocation
void Index();      //Indexed File Allocation
void Linked();     //Linked File Allocation



int main()
{
   int menu;
   printf("File Allocation Strategy\n");
   printf("\n1 : Sequential File Allocation\n");
   printf("2 : Indexed File Allocation\n");
   printf("3 : Linked File Allocation\n");
   printf("4 : Exit\n");
   printf("\nChoose File Allocation Technique\n");
   scanf("%d", &menu);
   if(menu==1)
   {
       system("cls");
       printf("Sequential File Allocation\n");
       Sequential();

     }




   if(menu==2)
   {
       system("cls");
       printf("Indexed File Allocation\n");
       Index();

   }
```

```
     if(menu==3)
    {
        system("cls");
        printf("Linked File Allocation\n");
        Linked();
    }

       if(menu==4)
        {
        return 0;
        }
}

//Sequential File Allocation

void Sequential()
{
    int n=0, i,j,a;
        char searchh[10];
        printf("\nEnter no. of files: ");
        scanf("%d", &n);
        fileTable ft[n];

        for(i=0; i<n; i++)
        {
            printf("\nEnter file name %d: ",i+1);
            scanf("%s", &ft[i].fN);
            printf("Enter starting block %d: ",i+1);
            scanf("%d", &ft[i].sB);
            printf("Enter number of blocks %d: ",i+1);
            scanf("%d", &ft[i].nB);
        }
        printf("\nEnter the file name to be searched :");
        scanf("%s", &searchh);

        for(i=0; i<n; i++)
        {
            if(strcmp(searchh, ft[i].fN) == 0)
            {
                printf("\nFile name\t Start Block \t No. of Blocks\t Blocks Occupied\n");
                printf("%s\t\t %d\t\t %d\t\t ", ft[i].fN, ft[i].sB, ft[i].nB);

                for(j=0; j<ft[i].nB; j++)
                {
                    if(j==0)
                    {
                        printf("%d", ft[i].sB+j);
                        continue;
                    }
                    printf(", %d", ft[i].sB+j);

                }
                break;
            }
        }

}

//Index File Allocation
```

```
void Index()
{
   char fileName[30];
      int num;
      printf("\nEnter no of files : ");
      scanf("%d",&num);
      Indexed fileBlocks[num];

      for(int i=0; i<num; i++)
      {
         printf("\nEnter file name %d : ",i+1);
         scanf("%s",&fileBlocks[i].name);
         printf("Enter no of blocks in file %d : ",i+1);
         scanf("%d",&fileBlocks[i].numB);
         printf("Enter the blocks of the file %d: ",i+1);
         for(int j=0; j<fileBlocks[i].numB; j++)
         {
            scanf("%d",&fileBlocks[i].Bnum[j]);
         }
      }
      printf("\nEnter the file name to be searched : ");
      scanf("%s",&fileName);
      for(int i=0; i<num; i++)
      {
         if(strcmp(fileName, fileBlocks[i].name)==0)
         {
            printf("\nFILE NAME\tNO OF BLOCKS\tBLOCKS OCCUPIED\n");
            printf("%s\t\t%d\t\t",fileBlocks[i].name,fileBlocks[i].numB);
            for(int j=0; j<fileBlocks[i].numB; j++)
            {
               if(j == 0)
```

```
                    {
                        printf("%d",fileBlocks[i].Bnum[j]);
                    }
                    else
                    {
                        printf(", %d",fileBlocks[i].Bnum[j]);
                    }
                }
                break;
            }
        }
}

//Linked File Allocation

void Linked()
{
    int n=0, i,j,a;
        char searchh[10];
        Block *fileBlocks;
        printf("\nEnter no. of files: ");
        scanf("%d", &n);
        nodeTable nt[n];
        for(i=0; i<n; i++)
        {
            printf("\nEnter file name %d: ",i+1);
            scanf("%s", nt[i].name);
            printf("Enter number of blocks %d: ",i+1);
            scanf("%d", &nt[i].nob);
            nt[i].sb = (block*) malloc(sizeof(Block));
            fileBlocks = nt[i].sb;
            printf("Enter the blocks of the file: ");
            fileBlocks->next = NULL;

            for(j=0; j<nt[i].nob; j++)
            {
                fileBlocks->next = (Block*) malloc(sizeof(Block));
                scanf("%d", &fileBlocks->bno);
                fileBlocks = fileBlocks->next;

            }
            fileBlocks->next = NULL;

        }
        printf("\nEnter the file name to be searched : ");
        scanf("%s", searchh);
        for(i=0; i<n; i++)
        {
            if(strcmp(searchh, nt[i].name) == 0)
            {
                printf("\nFile name\t No. of Blocks\t Blocks Occupied\n");
                printf("%s\t\t %d\t\t ", nt[i].name, nt[i].nob);
                fileBlocks = nt[i].sb;
                for(j=0; j<nt[i].nob; j++)
                {
                    if(j == 0){
                        printf("%d", fileBlocks->bno);
                        fileBlocks = fileBlocks->next;
                        continue;
                    }
                    printf("-> %d", fileBlocks->bno);
                    fileBlocks = fileBlocks->next;
                }
                break;
            }



      }
  }
```

Solving Methodology

In the file allocation strategy program, I used the C++ programming language. First I have to declare some structures for the whole program, named fileTable, Block, nodeTable, Indexed. Then I create some user define functions by the prototype for a particular allocation strategy. In the main function, I created the menu and used a switch-case for switching between functions. I made three functions, those are: Sequential, Index, and Linked. Detailed descriptions of those functions are given below:

Sequential:

- At first, I declared an array of fileTable structure which contains a character type array for the name of the block and two integers for starting block and number of blocks. I also declared a character type array for searching the block by name and an integer for the number of files.
- I took input from the user using the scanf function through a for loop. I also took the searching block from the user using the scanf function.
- Then I loop through all the files and compared its name by searched file name using strcmp. If the name matches then finally I printed all the blocks sequentially using another for loop.

Index:

- For the Index function, I declared an array of Indexed structure which contains a character type array for the name of the blocks, an integer for the number of blocks, and another integer type array to store the blocks number. I also declared a character type array for searching the block by name and an integer for the number of files.
- I took input from the user using the scanf function through a for loop. I also took the searching block from the user using scanf.

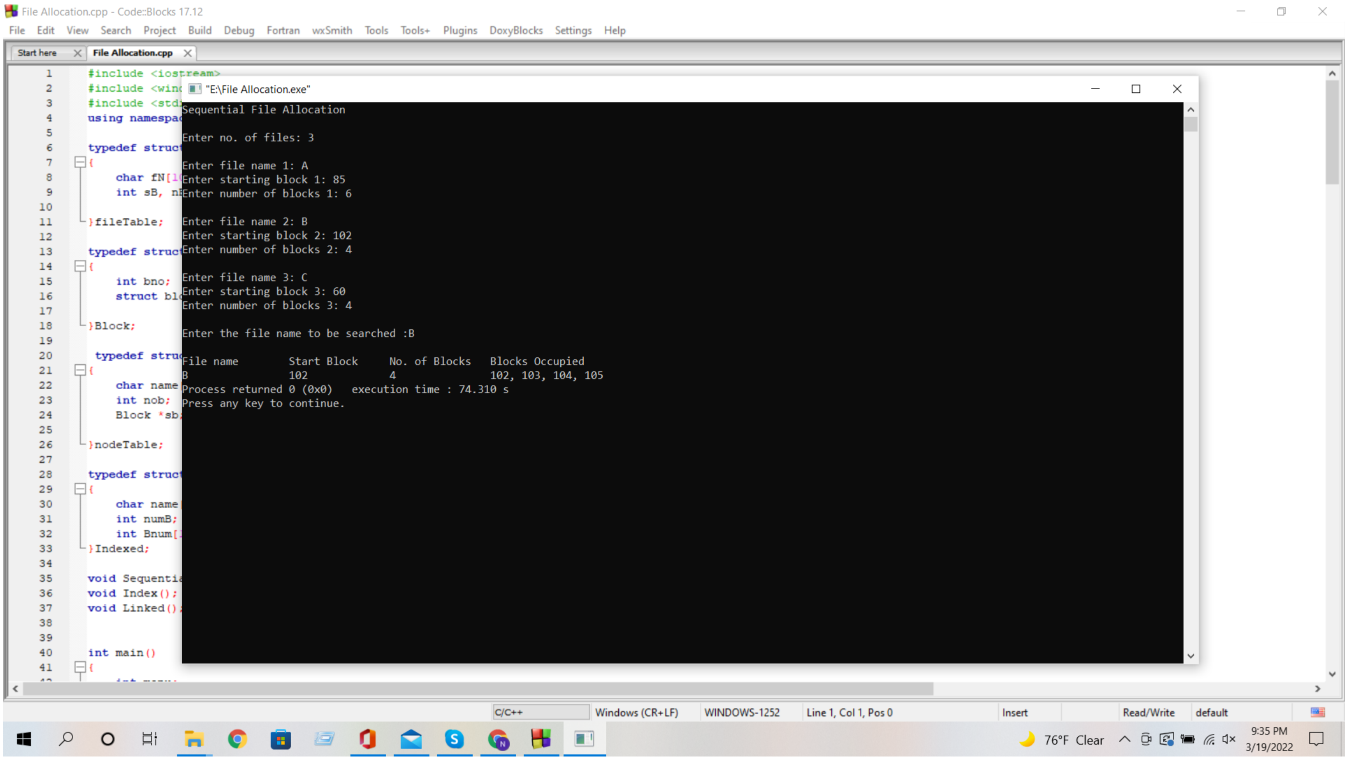
- Then I loop through all the files and compared its name by searched file name using strcmp. If the name matches then finally I printed all the block numbers that are stored in the particular structure index.

Linked:

- For the Linked function, I declared an array of nodeTable structure which contains a character type array for the name of the blocks, an integer for the number of blocks, and a linkedlist structure that store the blocks' number dynamically. I also declared a character type array for searching the block by name, an integer for the number of files, and a linkedlist pointer to point to the address of the block's number.
- I took input from the user using the scanf function through a for loop. I also took the searching block from the user using scanf.
- Then I loop through all the files and compared its name by searched file name using strcmp. If the name matches then finally I printed all the blocks by their address using linkedlist

Output Analysis

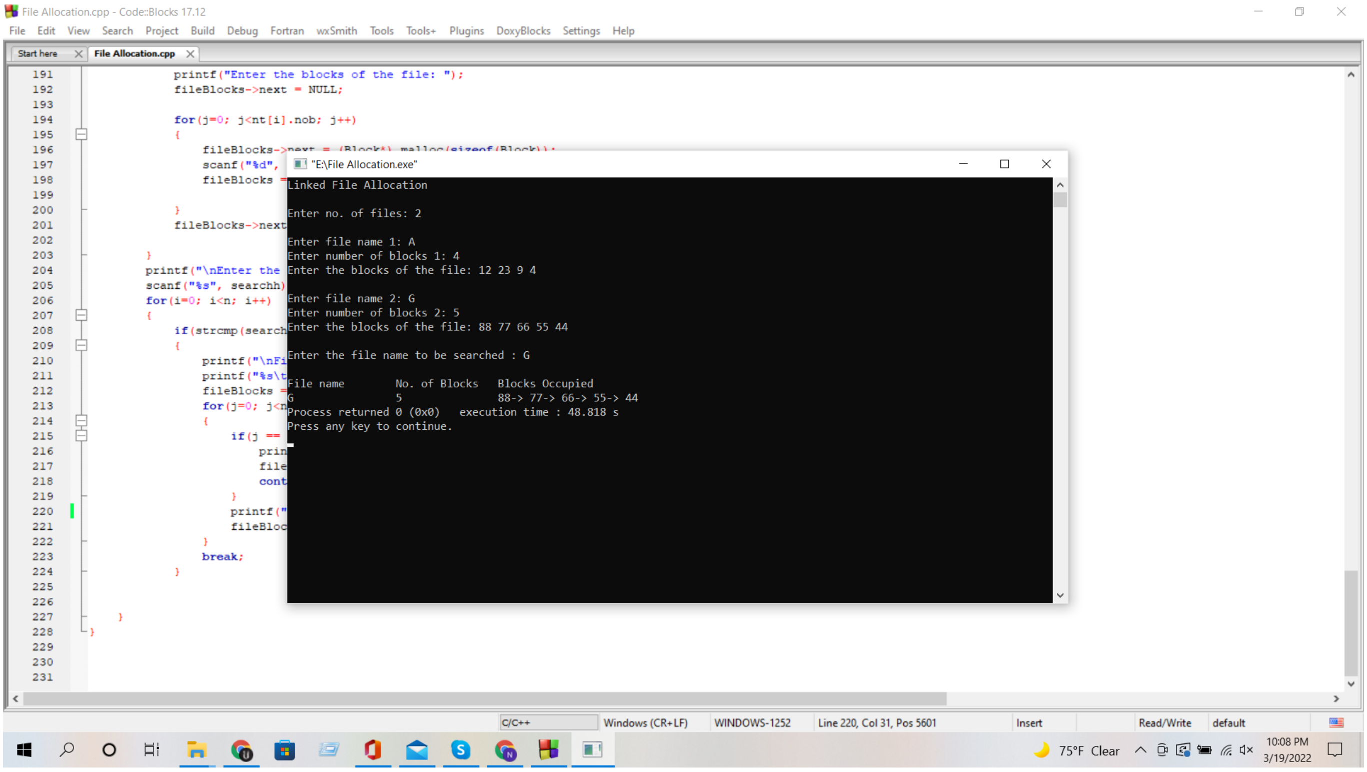
Sequential File Allocation



```
#include <iostream>
#include <wind
#include <stdi
using namespac

typedef struct
{
    char fN[1
    int sB, n

}fileTable;

typedef struct
{
    int bno;
    struct bl

}Block;

 typedef struc
{
    char name
    int nob;
    Block *sb;

}nodeTable;

typedef struct
{
    char name
    int numB;
    int Bnum[
}Indexed;

void Sequenti
void Index();
void Linked()


int main()
{
```

```
Sequential File Allocation

Enter no. of files: 3

Enter file name 1: A
Enter starting block 1: 85
Enter number of blocks 1: 6

Enter file name 2: B
Enter starting block 2: 102
Enter number of blocks 2: 4

Enter file name 3: C
Enter starting block 3: 60
Enter number of blocks 3: 4

Enter the file name to be searched :B

File name        Start Block     No. of Blocks   Blocks Occupied
B                102             4               102, 103, 104, 105
Process returned 0 (0x0)   execution time : 74.310 s
Press any key to continue.
```

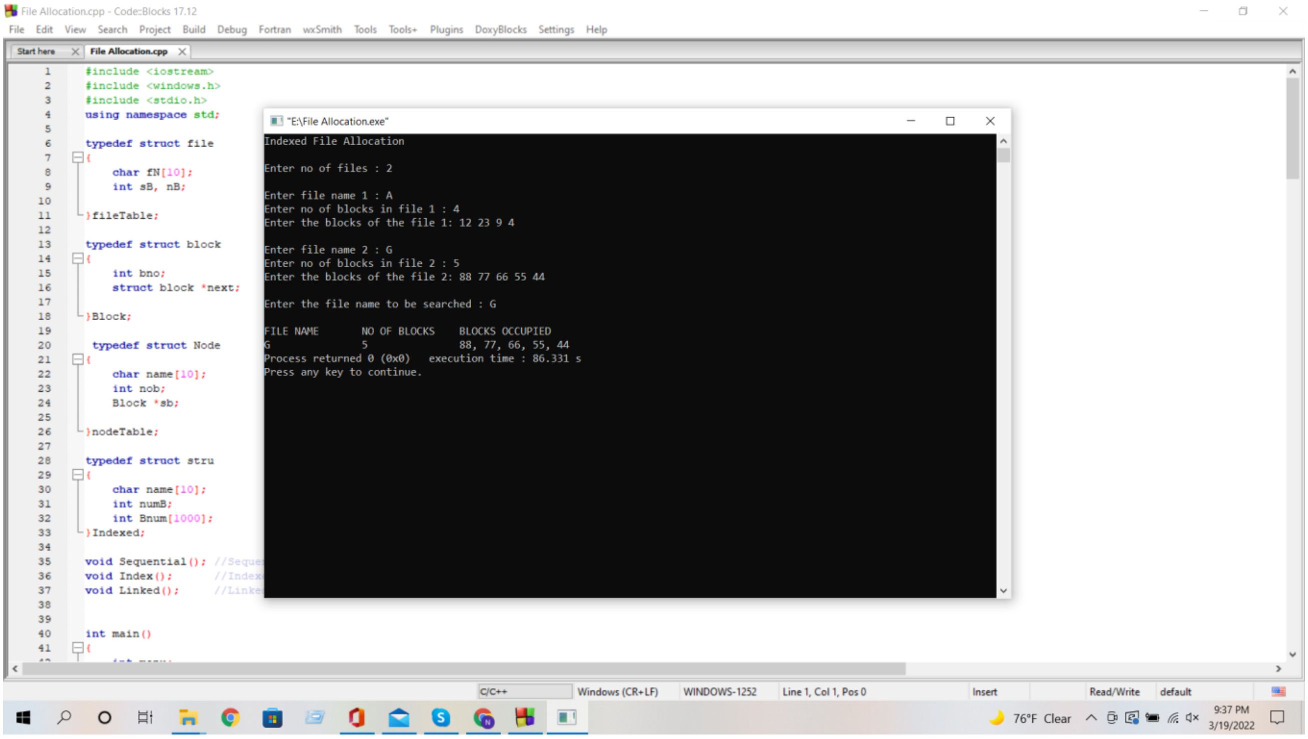
Linked File Allocation


File Allocation.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Start here
File Allocation.cpp
"E:\File Allocation.exe"
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 220, Col 31, Pos 5601
Insert
Read/Write
default
75°F Clear
10:08 PM
3/19/2022

```
            printf("Enter the blocks of the file: ");
            fileBlocks->next = NULL;

            for(j=0; j<nt[i].nob; j++)
            {
                fileBlocks->next = (Block*) malloc(sizeof(Block));
                scanf("%d",
                fileBlocks =

            }
            fileBlocks->next

        }
        printf("\nEnter the
        scanf("%s", searchh)
        for(i=0; i<n; i++)
        {
            if(strcmp(search
            {
                printf("\nFi
                printf("%s\t
                fileBlocks =
                for(j=0; j<n
                {
                    if(j ==
                        prin
                        file
                        cont
                    }
                    printf("
                    fileBloc
                }
                break;
            }


    }
}
```

```
Linked File Allocation

Enter no. of files: 2

Enter file name 1: A
Enter number of blocks 1: 4
Enter the blocks of the file: 12 23 9 4

Enter file name 2: G
Enter number of blocks 2: 5
Enter the blocks of the file: 88 77 66 55 44

Enter the file name to be searched : G

File name        No. of Blocks   Blocks Occupied
G                5               88-> 77-> 66-> 55-> 44
Process returned 0 (0x0)   execution time : 48.818 s
Press any key to continue.
```

Index File Allocation



```
#include <iostream>
#include <windows.h>
#include <stdio.h>
using namespace std;

typedef struct file
{
    char fN[10];
    int sB, nB;

}fileTable;

typedef struct block
{
    int bno;
    struct block *next;

}Block;

 typedef struct Node
{
    char name[10];
    int nob;
    Block *sb;

}nodeTable;

typedef struct stru
{
    char name[10];
    int numB;
    int Bnum[1000];
}Indexed;

void Sequential(); //Seque
void Index();      //Index
void Linked();     //Linke


int main()
{
```

```
Indexed File Allocation

Enter no of files : 2

Enter file name 1 : A
Enter no of blocks in file 1 : 4
Enter the blocks of the file 1: 12 23 9 4

Enter file name 2 : G
Enter no of blocks in file 2 : 5
Enter the blocks of the file 2: 88 77 66 55 44

Enter the file name to be searched : G

FILE NAME       NO OF BLOCKS    BLOCKS OCCUPIED
G               5               88, 77, 66, 55, 44
Process returned 0 (0x0)   execution time : 86.331 s
Press any key to continue.
```

Conclusion

The file allocation strategies has been solved by following the mentioned methodology above.