Lab 02

Data Structures

BS DS Fall 2024 Morning/Afternoon

Task 1

Define a function which prints the Row-Major based ND-array formula against a given number of dimensions.
For Example;

If the function is called as “printND(3)” then the following should be displayed on console:

$I_1U_2U_3 + I_2U_3 + I_3$

for “printND(1)”

I_1

for “printND(2)”

$I_1U_2 + I_2$

Where I_1 , I_2 , I_3 , ..., I_N represents the index set and
U_1 , U_2 , U_3 , ..., U_N represents the dimension set.

Task 2

Requirements:
Even though the multidimensional array is provided as a standard data object in C++, it is often useful to define your own class for multidimensional arrays. This gives a more robust class that:

- Does require the index set in each dimension to start at any number (€ Set of Integers).
- Selects range of each dimension of the array during runtime.
- Provide mechanism to determine the size of array.

No specific data structure is being told to you for this task so intelligently do a proper structure selection yourself and when feels confident then also discuss your structure with Instructor before start coding.

Sample Program Run

```
int dim-size[3]={5,3,10};

NDArray arr(3, dim-size);

int index-set[3]={4,2,8};
cout<<arr.calculateIndex( indexset )<<endl;

arr.setValue( index-set , val);

cout<<arr.getValue(index-set);
```

Task 3: Deals with buffered writing!

Objective:

- The main objective of this is to experience the effect on performance (time) with the use of buffered reading/writing.
- Getting the basic concept of indexing and its advantage on performance (time).
- Hands on STL.

Prerequisite Setup:

1. **Define a struct 'Student' as follows:**

```
struct Student
{
        int roll;
        char name[30];
        Student():roll(0)
        {
                strcpy(name,"none");
        }
};
```

2. **The following code stores 10 million records of 'Student' in a file in binary mode pass N = 10000000**

```
void addToStudentUnBuffered(int N)
{
        ofstream  ofs("studentdatabase.txt",ios::binary|ios::out);
        Student s;
        for (int i=1; i<N; i++)
        {
                s.roll = i;
                ofs.write((char*)(&s),sizeof(Student));
        }
        ofs.close();
}
```

3. **Write the following code in main: [Required header files stdio.h and windows.h]**

```

SYSTEMTIME systime;
cout<<"\nWriting Records to File one by one";
GetLocalTime(&systime);
cout<<endl<<systime.wHour<<":"<<systime.wMinute<<":"<<systime.wSecond<<":"<<systi
me.wMilliseconds;
addToStudentUnBuffered();
GetLocalTime(&systime);
cout<<endl<<systime.wHour<<":"<<systime.wMinute<<":"<<systime.wSecond<<":"<<systi
me.wMilliseconds;
```

Can you improve the performance of addToStudentUnBuffered ?
Idea is to go for buffered writing. Name your function as addToStudentBuffered

Task 4: Deals with buffered reading!

Write a function which display all the Student records stored in "studentdatabase.txt" in Task-3.
[Maybe you noticed or not: its 343 MB file]

Note: Write two versions of display all records as follows:

➔ readAllRecordsUnBuffered() //:It reads records from file one by one i.e. unbuffered.
➔ readAllRecordsBuffered() //:It reads records from file in a buffer.

Then observe the response time of both functions by getting the system time.