# CS201 (LAB)

**Practice Program: 1 (Theme: Learn to write Makefile)**

Write a C/C++ program to read a text file and count the number of occurances of the words present in the file. Display the words in sorted order (increasing order of their counts). Your program should be structured as follows:

readFile.cpp: file handling file
  -- functions to read a text file

readFile.h: function declaration of the functions present in readFile.cpp

charHandle.cpp: string handling file
  -- any function which performs string manipulation

1. extracting strings from the buffer read from the read function define in readFile.cpp
2. counting of the words

charHandle.h: function declaration

wordSort.cpp: sort handling file.
  -- all the sorting related function should be placed under this file.
wordSort.h: sort functions declarations
myProg.cpp: the file which contains only main() function.

Note: you can give the file name of your choice. If required, you can also create more files and more functions. Every module/function should be as small as possible i.e., each module should perform smallest possible task.

All the user's data type declarations/ function declarations should be placed under *.h files. The definition of the functions should be placed under *.cpp files. The main program should have only the main function.

**You need to strictly follow the directory structure as explained in the theory class.**

**Practice Problem 2 – (Theme: class handling)**
**Dictionary Search:** This assignment involves writing some classes, constructors, destructors, memory allocation, file I/O (you can use C file handling). Write a c++ program to read a dictionary entries from a file and perform search operations. Your program should have two classes: Word and Dictionary.

Your program should adhere to the following specifications:

1. The Word class should handle word level operation such as creation of a word.
2. The Dictionary class should handle dictionary level operation such as dictionary creation, insert a word in the dictionary, search a word in the dictionary etc.
3. You are encouraged to add additional supporting functions (if necessary)
4. The Word class should dynamically allocate memory for each word to be stored in the dictionary.
3. The Dictionary class should contain an array of pointers to Word. The memory for this array must be dynamically allocated. Read the dictionary words in from a dictionary file. The size of the pointer should grow dynamically as you read the file in. Start with an array size of 8, When that array is filled, double the array size, copy the original 8 words to the new array, free the old array and continue.
4. Your program should take list of words one after another and check the presence of the word in the dictionary. If found, output the corresponding dictionary meaning as specified in the file.

**Sample dictionary file: dict.txt**
```
about : Approximately; nearly.
class : A level of academic development, as in an elementary or secondary school.
class : A division based on quality, rank, or grade.
.....
```

**Suggested Class Structure**
```
class Word
{
  char* word;
  char* meaning;
  ....
public:
  Word(....);
  ~Word();
  const char* const word() const;
};
class Dictionary
{
  Word** words;
  unsigned int sizeOfDict;
  void resize();
  void insert(char* word);
  void delete(char* word);
  ...
public:
  Dictionary(const char* filename);
  ~Dictionary();
  bool find(const char* word);
};
int main()
{ Dictionary dict("dict.txt");
  ...
  cin >> "inter a word to search" >> searchWord;
  while (searchWord != 0) {
       if (!dict.find(searchWord)) {
          cout << searchWord << " not found in the Dictionary\n";
       }
       else{ .......                    }
       cin >> "inter a word to search" >> searchWord;
  }
  return 0;
}
```

**Practice Problem 3: (Theme: Class handling)**
Books Store: Some of the characteristics of a book are the title, author(s), publisher, ISBN, price, and year of publication.

Design a class bookType that defines the book as an ADT.

1. Design a class Book that defines the book as an ADT.
2. Each Book should have title, author(s), publisher, ISBN, price, and year of publication as private members.
3. It should have book handling operations such as book_name(), book_title(), year_publiction(), book_isbn() etc. (you are encourage to add more)
4. Further, you should have another class name Store to maintain the store. A store has a collection on book object and the operations such as add_book, delete_book, find_book, number_of_books_in_store, sale and more (in necessary)
5. The store object should maintain the stock and sale status.
6. A store keeper should be able to add a new book

**Practice Problem 4 – (Theme: Overloading)**
4.1
Overload the function ***find*** in problem number 2 with multiple words i.e., allow multiple word search with single finctuion call

4.2
Write a program to create a complex number class and support complex number operation i.e., complex number +, -, *, ==, (all possible operators).

4.3
Using the complex number class defined above, write the operator overloading functions for "<<" (for output) and ">>"( for input). While overloading >>, enter the real and imaginary parts separately in different lines

or separated by space. While overloading <<, display the real part and imaginary part clearly mention.

4.4

Overload the add_book, delete_book functions in problem 3 by + and -. Also allow to merge two stores objects to a single Store object using the same + operator.

4.5

Overload the insert, delete functions in problem 2 by + and -.