Test Run Code:

```
#include<iostream>
#include<fstream>

using namespace std;

ofstream output;        //to write output a file

void match(string text, string pattern)
{
        int len_text= text.length();
        int len_pattern= pattern.length();

        cout<<"\n Positions of text \""<<text<<"\" , where pattern \""<<pattern<<"\" is
found:\n";

        //write output to output file
        output<<"\n Positions of text \""<<text<<"\" , where pattern \""<<pattern<<"\"
is found:\n";

        int i, j,flag=0;      //flag for identify whether pattern is found at lease
once

        //if pattern is null
        if(pattern==""){
                cout<<"\n\t---Pattern is empty. Nothing to find in Text---\n";
                output<<"\n\t---Pattern is empty. Nothing to find in Text---\n";
        }

        //if pattern is not null neive string matching start
        else{
                for(i=0;i<=len_text-len_pattern;i++){

                   for(j=0;j<len_pattern;j++){

                       //if (j+1) th position of pattern is not underscore or
                       //not equal to (i+j+1)th position of text
                       if(pattern[j]!=text[i+j] && pattern[j]!='_')
                            break;
                   }

                   //if pattern found in (i+1)th position in text
                   if(j==len_pattern){

                      //print position and pattern in text
                      cout<<"\n\t"<<i+1<<" - \'"<<text.substr(i,len_pattern)<<"\'\n";
                      output<<"\n\t"<<i+1<<" - \'"<<text.substr(i,len_pattern)<<"\'\n";

                      flag=1;                 //found pattern once
                   }
                }
        }

        //if pattern not found in text
        if(flag==0)
                        cout<<"\n\t---Pattern is not found in Text---\n";
                        output<<"\n\t---Pattern is not found in Text---\n";
}
```

```
//to read file and return line
string fileread(string filepath, int fileno){
        string line, file;
        file=filepath + to_string(fileno) + ".txt";    //prepare file name to read file
        ifstream infile;                               //file pointer
        infile.open (file.c_str());                    //open file to read
        getline(infile,line);                          //read line and store
        infile.close();                                //close file
        return line;
}

int main(){
        string pattern,text,outfile;    //declare to store user pattern and text string
        bool run=true;
        int i=1;
        int no=6;               //6 test cases
        char con;

        do{
                cout<<"\n ------------------TEST CASE "<<i<< "------------------";
                text=fileread("Test/Text/text",i);            //read i th text file
                pattern=fileread("Test/Pattern/pattern",i);   //read i th pattern file

                outfile= "Test/Output/patternmatch" + to_string(i) + ".output";
                                                 //prepare file name to write output
                output.open(outfile.c_str());   //open output file

                match(text,pattern);             //search pattern in text
                output.close();                  //close output file

                i++;

                if(i<=no){
                        cout<<"\n Continue? [Y/N]";
                        cin>>con;
                        if(con=='N' || con=='n')
                                run=false;
                }

        }while(run && i<=no);    //loop if user want continue and have test case to run

        return 0;
}
```

Application Code:

```
#include<iostream>

using namespace std;
```

```
void match(string text, string pattern)
{
        int len_text= text.length();
        int len_pattern= pattern.length();

        cout<<"\n Positions of text \""<<text<<"\" , where pattern \""<<pattern<<"\" is
found:\n";

        int i, j,flag=0;     //flag for identify whether pattern is found at lease once

        //if pattern is null
        if(pattern==""){
                cout<<"\n\t---Pattern is empty. Nothing to find in Text---\n";
        }

        //if pattern is not null neive string matching start
        else{
                for(i=0;i<=len_text-len_pattern;i++){

                   for(j=0;j<len_pattern;j++){

                       //if (j+1) th position of pattern is not underscore or
                       //not equal to (i+j+1)th position of text
                       if(pattern[j]!=text[i+j] && pattern[j]!='_')
                            break;
                   }

                   //if pattern found in (i+1)th position in text
                   if(j==len_pattern){

                      //print position and pattern in text
                      cout<<"\n\t"<<i+1<<" - \'"<<text.substr(i,len_pattern)<<"\'\n";

                      flag=1;                 //found pattern once
                   }
                }
        }

        //if pattern not found in text
        if(flag==0)
                        cout<<"\n\t---Pattern is not found in Text---\n";
}

int main(){
        string pattern,text;            //declare to store user pattern and text string

        cout<<" Enter TEXT: ";
        getline(cin, text);             //get line as text for search pattern

        cout<<"\n Enter PATTERN: ";
        getline(cin, pattern);          //get line as search pattern with underscores

        match(text,pattern);             //search pattern in text

        return 0;
}
```

Code Explanation:

Naive string matching algorithm is used in here. Rabin-Karp can not be used because hash values of pattern and corresponding part of text may not be same because of wildcard character whether it is a matching pattern. KMP and Boyer-Moore can be modified according to wildcard. But I thought that it will little harder than modify naive string algorithm. So I used Naive string algorithm because it can be modified easily to wildcard by going one by one through text to find pattern.

```
void match(string text, string pattern)
{
	int len_text= text.length();
	int len_pattern= pattern.length();

	cout<<"\n Positions of text \""<<text<<"\" , where pattern \""<<pattern<<"\" is found:\n";

	int i, j,flag=0;				//flag for identify whether pattern is found at lease once

	//if pattern is null
	if(pattern==""){
		cout<<"\n\t---Pattern is empty. Nothing to find in Text---\n";
	}
	//if pattern is not null neive string matching start
	else{
		for(i=0;i<=len_text-len_pattern;i++){

			for(j=0;j<len_pattern;j++){

				//if (j+1) th position of pattern is not underscore or
				//not equal to (i+j+1)th position of text
				if(pattern[j]!=text[i+j] && pattern[j]!='_')
					break;
			}

			//if pattern found in (i+1)th position in text
			if(j==len_pattern){

				//print position and pattern in text
				cout<<"\n\t"<<i+1<<" - \'"<<text.substr(i,len_pattern)<<"\'\n";

				flag=1;				//found pattern once
			}
		}
	}

	//if pattern not found in text
	if(flag==0)
			cout<<"\n\t---Pattern is not found in Text---\n";
}
```

Text and pattern that wants to be search are passed to **match()** function. In application code they will be user inputs and in test run code they will be contain of files. Length of pattern and text are found. If pattern is null, then there is nothing to find. Otherwise pattern is searched in text by neive string matching algorithm by two loops. But there is one difference from normal naive string algorithm. In naive string algorithm, each pattern character in pattern is equal to corresponding text character in the shift is called as pattern is found in a shift. But in here it is developed as each pattern character in pattern is equal to corresponding text character in the shift or pattern character is wildcard character, is called as pattern is found in a shift. Otherwise break inner loop to go to next shift as below.

```
if(pattern[j]!=text[i+j] && pattern[j]!='_')
        break;
```

If pattern is found in a shift, position and matched pattern are displayed as below.

```
cout<<"\n\t"<<i+1<<" - \'"<<text.substr(i, len_pattern)<<"\'\n";
```

If pattern is not found in text at least once, it will output that pattern is not found.

In application code user input of pattern and text get in **main()** function as below.

```
int main(){
     string pattern,text;            //declare to store user pattern and text string

     cout<<" Enter TEXT: ";
     getline(cin, text);            //get line as text for search pattern

     cout<<"\n Enter PATTERN: ";
     getline(cin, pattern);         //get line as search pattern with underscores

     match(text,pattern);                    //search pattern in text

     return 0;
}
```

Those user inputs are stored in text and pattern variables and they passed to **match()** function to search pattern in text.

In test run code **fstream** header files is added to read and write file for testing test cases and its outputs.

```
string fileread(string filepath, int fileno){
     string line, file;
     file=filepath + to_string(fileno) + ".txt";   //prepare file name to read file
     ifstream infile;                              //file pointer
     infile.open (file.c_str());                  //open file to read
     getline(infile,line);                       //read line and store
     infile.close();                            //close file
     return line;
}
```

We have several test file sets to run. Each set has 3 files that have stored text and pattern and last one to store output. We have to read each text and pattern file set. **fileread()** function is called with file path and file no. (whether text file or pattern file path. Eg: Test/Text/text) and file no to return content of file(text or pattern). Full file path will be prepared by concatenation of file path and file no. and ".txt" extension. To open that file to read, we declare **infile** pointer in **ifstream.** File open will be done by **infile.open (file.c_str())** get content of file by **getline(infile,line)**. After close file content will be returned.

```
int main(){
    string pattern,text,outfile;    //declare to store user pattern and text string
    bool run=true;
    int i=1;
    int no=6;              //6 test cases
    char con;

    do{
        cout<<"\n ------------------TEST CASE "<<i<< "------------------";
        text=fileread("Test/Text/text",i);          //read i th text file

        pattern=fileread("Test/Pattern/pattern",i);   //read i th pattern file

        outfile= "Test/Output/patternmatch" + to_string(i) + ".output";
                                        //prepare file name to write output
        output.open(outfile.c_str());   //open output file

        match(text,pattern);          //search pattern in text
        output.close();              //close output file

        i++;

        if(i<=no){
            cout<<"\n Continue? [Y/N]";
            cin>>con;
            if(con=='N' || con=='n')
                run=false;
        }

    }while(run && i<=no);    //loop if user want continue and have test case to run

    return 0;
}
```

Still we have 6 test cases. In main method, each test run will be run in loop until user stop run or test runs are over. We have to prepare file name to store output in corresponding file. That is same as prepare file name to read files previously.(**"Test/Output/patternmatch" + to_string(i) + ".output").** To open that file to write, we declare **output** pointer in o**fstream** in global scope because **output** is used in **main()** function and **match()** function in test run code. Each output of test case is print in terminal as well as write to the output file. Print is happened by **cout** (**cout<<"\n\t---Pattern is empty. Nothing to find in Text---\n").** Write ouput to the file is also happened as print.(**output<<"\n\t---Pattern is empty. Nothing to find in Text---\n"**)