CONTENTS

1. Introduction

1.1 Problem Definition

The existing manual Voting system consumes more time for Vote Casting. Voter has to wait for vote polling station to vote for a right candidate. The election officers has to be check the voter , this voter can vote in this booth then check voterID present in voters list of booth those are information will be present then the voter can vote in that booth. The voter had to stand in the queue to cast his vote. All the work is done in paper ballot so it is very hard to locate a particular candidates, some voters cast their votes for all candidates. To overcome of all these problems we have to implement a web application, which is helpful for voting from anywhere.

1.2 About Project

The objective of the system is a replacement of the traditional system that is in existence. This smart system reduces the time for voting and also the system is reliable, and faster. In this system, the voter userID can be entered to cast votes to the desired candidate. Database maintained by this system usually contains the Voters information, Banned Voters, Candidate information, Deleted Votes, Previous Election Data and The final result of total votes and percentage of votes casted by total voters.

2. Project Analysis

2.1 Existing System

The voting system currently being used by the association is a paper based system, in which the voter simply picks up ballots sheets from electoral officials, tick off who they would like to vote for, and then cast their votes by merely handing over the ballot sheet back to electoral official.
The electoral officials gather all the votes being cast into a ballot box. At the end of the elections, he electoral officials converge and count the votes cast for each candidate and determine the winner of each election category.

2.2 Proposed System

Here we are proposing an web application for voting process that is Mini Voting System. This voting system will manages the voter's details, Candidate details. The main feature of the project includes voters information and candidate information, voter can login and use his/her voting rights. There is a separate admin and students panel.

Here Students are referred to as Voters and Voter Panel as Student Panel.

It gives Result on the basis of votes casted by voters and shows all the required data and stats separately.

The system can manage the information data very efficiently. The proposed system is more reliable, faster, accurate and easy to handle compared to existing manual system. It helps to computerize everything and reducing the errors as compare to manual voting system

3. System Requirements

3.1 Hardware Requirements

- Processor: Pentium/Intel Core/AMD Ryzen
- RAM: 4GB
- Hard Disk: 1TB
- Speed: 1.1GHz

3.2 Software Requirements

- Operating System: Windows
- Back-end Technology: C Programming
- Supporting Tools: VS Code
- Type: Offline
- Compiler Used: GCC Compiler

4. Tools and Technologies Used

4.1 Introduction to C Programming Language

C programming language is a procedural and general-purpose programming language. It is fast and simple to learn and implement. It was developed by Dennis Ritchie in the year of 1972.

The c programming language offers various features and functionalities to the programmers. It includes low-level memory access, simple syntax, and a clean style. It makes the C programming language suitable for system programming like compiler development and operating system development.

After C, several programming languages like PHP, JavaScript, Java, and many others have borrowed the syntax and the features directly or indirectly from the C language.

If someone wants to learn a programming language then he/she should start learning the C programming language in the first place. C is the base for programming.

The C language is a high-level, general-purpose programming language. It provides a straightforward, consistent, powerful interface for programming systems. That's why the C language is widely used for developing system software, application software, and embedded systems.

The C programming language has been highly influential, and many other languages have been derived from it. For example, C++ and Java are two popular modern dialects of C.

And C is an excellent choice for system programming, for example, developing operating systems, compilers, and network drivers. Despite its popularity, C is not without its criticisms. Some have argued that its syntax could be more complex and easier to learn, while others have noted its lack of standardization as a significant issue. Nevertheless, C remains a widely used and influential language and will probably continue for many years.

4.1.1 What is C?

It is a very powerful and general-purpose language used in programming. We can use C to develop software such as databases, operating systems, compilers, and many more. This programming language is excellent to learn for beginners in programming.

Prerequisites of using the C Language

Before you proceed ahead with the understanding and learning of the C language, you must have a basic grasp of the terminologies of computer programming. If you understand the basics of any computer programming language, then you will be able to effectively understand the C language.

Why Learn C Programming Language?

This language is a must for those working professionals as well as students who want to become established software engineers. Here are some of the key reasons why you must learn the C language for the domain of software development:

- This language helps users comprehend a computer's internal architecture. It assists you in knowing how a computer would store information within and retrieve it.
- Learning other programming languages becomes easier after learning C, such as Python, Java, etc.
- A programmer who is well-versed with the C language gets opportunities to work on various open-source projects. For instance, some of the very popular projects (open-source) have been written using the C programming language, such as Python interpreter, Linux kernel, SQLite database, and many more.

Benefits of C Programming Language

- It is a structured language.
- The C language is very easy to understand and learn.
- The C language generates very efficient programs.
- It helps you handle various low-level activities.
- The compilation of C programs can occur on various computer programs.

Facts About the C Language

- The C language came into existence for writing an OS known as the UNIX operating system.
- This language is the successor of the B language, which came into existence in the early 1970s.
- The ANSI (American National Standard Institute) formalized this language in 1988.
- The creators of this language have totally written the UNIX OS in C.
- As of today, the C language is the most popular and widely used programming language.
- Programmers have implemented a majority of the futuristic, avant-garde tools and software using the C language.

4.1.2 History of C Programming Language

The C programming language was created at Bell Laboratories in the early 1970s, mainly by Ken Thompson and Dennis Ritchie. For the UNIX operating system, which at the time required applications to be written in assembly language, programmers needed a more user-friendly set of instructions. Assembly programmes, which communicate directly with a computer's hardware, are lengthy and complex to debug, and adding new functionality requires a lot of time and effort.

Thompson's first high-level language was named B after the BCPL system programming language on which it was built. Thompson rewrote B to better match the demands of the modern time, better system hardware after Bell Labs purchased a Digital Equipment Corporation (DEC) UNIX system model PDP-11. As a result C, the B's successor, was created. By 1973, C had matured to the point that it could be used to rewrite the UNIX operating system.

Other programmers needed documentation that detailed how to use C before it could be utilized effectively outside of Bell Labs. In 1978, Brian Kernighan and Dennis Ritchie's book "The C Programming Language," sometimes known as K&R or the "White Book" by C enthusiasts, became the canonical source for C programming. The second edition of K&R, first published in 1988, is still commonly accessible as of this writing. Based on the book, the first, pre-standard version of C is known as K&R C.

Throughout the 1980s, C developers sought to build standards for the language in order to prevent others from developing their own dialects. The American National Standards Institute (ANSI) standard X3.159-1989 became the official U.S. standard for C in 1989. In 1990, the International Organization for Standardization (ISO) issued the ISO/IEC 9899:1990 standard. These standards, as well as their later updates, are referenced in C versions after K&R. (C89, C90 and C99).

The 1980s saw a surge in operating system development, with C and its use in UNIX being only such instances. Despite its advancements over its predecessors, C was still difficult to use for creating larger software programmes. As computers got more powerful, there was a growing demand for a more user-friendly programming environment. This desire pushed programmers to use C to create their own compilers and, as a result, new programming languages. These new languages may make it easier to code complex operations with many moving elements. For example, object-oriented programming, a programming method that maximizes a programmer's ability to reuse code, was eased by languages like C++ and Java, both of which were derived from C.

4.1.3 Commands and Their Functions

Commands	Functions
#include	This is the main header file preprocessor function, which preprocesses standard input and output header files from the C library repository, such as stdio.h, before compiling the programme.
int main()	This C statement, like most programming languages, is the main function, which is the point where the programme execution begins. Once the primary main () has been executed, all other methods and functions are performed.
{	Curly braces are a type of bracket that can be seen in any computer language, not just C. This represents the start of a method or function definition.
/* explanation of C code */	The text inside the /* and */ tags will be considered as comments and will not be executed or compiled. This is to provide the coder with a clear knowledge of the code and its application or use.
Printf	The output is printed to the console screen using this C command.
return 0	This C function returns 0 after terminating the C programme or main function.
}	The function or method block is closed with these curly braces.
//	These are called single line comments, and they are utilized not only in the C programming language but in others as well.
return	The output of the code execution is returned using this function.
Scanf	The user data is taken from the usual console terminal window using this C function.

4.1.4 Variables in C

Variable is basically nothing but the name of a memory location that we use for storing data. We can change the value of a variable in C or any other language, and we can also reuse it multiple times. We use symbols in variables for representing the memory location- so that it becomes easily identifiable by any user.

Use of the Variables in C

Variables are the storage areas in a code that the program can easily manipulate. Every variable in C language has some specific type- that determines the layout and the size of the memory of the variable, the range of values that the memory can hold, and the set of operations that one can perform on that variable.

The name of a variable can be a composition of digits, letters, and also underscore characters. The name of the character must begin with either an underscore or a letter. In the case of C, the lowercase and uppercase letters are distinct. It is because C is case-sensitive in nature. Let us look at some more ways in which we name a variable.

Rules for Naming a Variable in C

We give a variable a meaningful name when we create it. Here are the rules that we must follow when naming it:

1. The name of the variable must not begin with a digit.

2. A variable name can consist of digits, alphabets, and even special symbols such as an underscore (_).

3. A variable name must not have any keywords, for instance, float, int, etc.

4. There must be no spaces or blanks in the variable name.

5. The C language treats lowercase and uppercase very differently, as it is case sensitive. Usually, we keep the name of the variable in the lower case.

Data Type of the Variable

We must assign a data type to all the variables that are present in the C language. These define the type of data that we can store in any variable. If we do not provide the variable with a data type, the C compiler will ultimately generate a syntax error or a compile-time error.

The data Types present in the C language are float, int, double, char, long int, short int, etc., along with other modifiers.

Types of Primary/ Primitive Data Types in C Language

Type of Variable	Name	Description	Uses
char	Character	It is a type of integer. It is typically one byte (single octet).	We use them in the form of single alphabets, such as X, r, B, f, k, etc., or for the ASCII character sets.

int	Integer	It is the most natural size of an integer used in the machine.	We use this for storing the whole numbers, such as 4, 300, 8000, etc.
float	Floating-Point	It is a floating-point value that is single precision.	We use these for indicating the real number values or decimal points, such as 20.8, 18.56, etc.
double	Double	It is a floating-point value that is double precision.	These are very large-sized numeric values that aren't really allowed in any data type that is a floating-point or an integer.
void	Void	It represents that there is an absence of type.	We use it to represent the absence of value. Thus, the use of this data type is to define various functions.

Declaration of Variable in C

Declaring a variable provides the compiler with an assurance that there is a variable that exists with that very given name. This way, the compiler will get a signal to proceed with the further compilation without needing the complete details regarding that variable.

The variable definition only has a meaning of its own during the time of compilation. The compiler would require an actual variable definition during the time of program linking.

The declaration of variables is useful when we use multiple numbers of files and we define the variables in one of the files that might be available during the time of program linking. We use the *extern* keyword for declaring a variable at any given place. Though we can declare one variable various times in a C program, we can only define it once in a function, a file, or any block of code.

Classification of Variables in C

The variables can be of the following basic types, based on the name and the type of the variable:

- **Global Variable**: A variable that gets declared outside a block or a function is known as a global variable. Any function in a program is capable of changing the value of a global variable. It means that the global variable will be available to all the functions in the code. Because the global variable in c is available to all the functions, we have to declare it at the beginning of a block. Explore, Global Variable in C to know more.
  Example,
  int value=30; // a global variable
  void function1(){
  int a=20; // a local variable
  }
- **Local Variable**: A local variable is a type of variable that we declare inside a block or a function, unlike the global variable. Thus, we also have to declare a local variable in c at the beginning of a given block.
  Example,
  void function1(){

int x=10; // a local variable
}

- **Automatic Variable**: Every variable that gets declared inside a block (in the C language) is by default automatic in nature. One can declare any given automatic variable explicitly using the keyword *auto*.
  Example,
  void main(){
  int a=80; // a local variable (it is also automatic variable)
  auto int b=50; // an automatic variable
  }
- **Static Variable**: The static variable in c is a variable that a user declares using the *static* keyword. This variable retains the given value between various function calls.
  Example, void function1(){

  int a=10; // A local variable

  static int b=10; // A static variable

  a=a+1;

  b=b+1;

  printf("%d,%d",a,b);

  }

  If we call this given function multiple times, then the local variable will print this very same value for every function call. For example, 11, 11, 11, and so on after this. The static variable, on the other hand, will print the value that is incremented in each and every function call. For example, 11, 12, 13, and so on.

- External Variable: A user will be capable of sharing a variable in multiple numbers of source files in C if they use an external variable. If we want to declare an external variable, then we need to use the keyword *extern*.
  Syntax, extern int a=10;// external variable (also a global variable)

Rvalues and Lvalues in C

There are two types of expressions in the C language:

rvalue − This term refers to the data value that gets stored at some type of memory address. The rvalue is basically an expression that has no value assigned to it. It means that an rvalue may appear on the RHS (right-hand side), but it cannot appear on the LHS (left-hand side) of any given assignment.

lvalue − lvalue expressions are the expressions that refer to any given memory location. The lvalue can appear as both- the RHS as well as the LHS of any assignment.

A variable is an lvalue. Thus, it may appear on the LHS of an assignment as well, along with the RHS. The numeric literals are rvalues. Thus, these might not be assigned. Thus, these can't appear on the LHS. Here are two statements- one valid and invalid.

4.1.5 Data Types in C

Just like the name suggests, here, data types refer to the type of data that we are using in a C program. Whenever we utilise a data type in a C program, we define the variables or functions used in it. We do so because we must specify the type of data that is in use, so that the compiler knows exactly what type of data it must expect from the given program.

Purpose of Data Types in C

Data types used in C language refer to an extensive system that we use to declare various types of functions or variables in a program. Here, on the basis of the type of variable present in a program, we determine the space that it occupies in storage, along with the way in which the stored bit pattern will be interpreted.

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

Types of Data Types in C

Here are the five major categories into which data types are divided in C language:

Data Type	Example of Data Type
Basic Data Type	Floating-point, integer, double, character.
Derived Data Type	Union, structure, array, etc.
Enumerated Data Type	Enums
Void Data Type	Empty Value
Bool Type	True or False

The basic data types are also known as the primary data types in C programming.

Primary Data Types in C

Here are the five primitive or primary data types that one can find in C programming language:

1. Integer – We use these for storing various whole numbers, such as 5, 8, 67, 2390, etc.

2. Character – It refers to all ASCII character sets as well as the single alphabets, such as ‘x’, ‘Y’, etc.

3. Double – These include all large types of numeric values that do not come under either floating-point data type or integer data type. Visit Double Data Type in C to know more.

4. Floating-Point – These refer to all the real number values or decimal points, such as 40.1, 820.673, 5.9, etc.

5. Void – This term refers to no values at all. We mostly use this data type when defining the functions in a program.

Range of Values of C Data Type

Data Type	Format Specifier	Minimal Range	Typical Bit Size
unsigned char	%c	0 to 255	8
char	%c	-127 to 127	8
signed char	%c	-127 to 127	8
int	%d, %i	-32,767 to 32,767	16 or 32
unsigned int	%u	0 to 65,535	16 or 32
signed int	%d, %i	Same as int	Same as int 16 or 32
short int	%hd	-32,767 to 32,767	16
unsigned short int	%hu	0 to 65,535	16
signed short int	%hd	Same as short int	16
long int	%ld, %li	-2,147,483,647 to 2,147,483,647	32
long long int	%lld, %lli	-(263 – 1) to 263 – 1 (It will be added by the C99 standard)	64
signed long int	%ld, %li	Same as long int	32
unsigned long int	%lu	0 to 4,294,967,295	32
unsigned long long int	%llu	264 – 1 (It will be added by the C99 standard)	64
float	%f	1E-37 to 1E+37 along with six digits of the precisions here	32
double	%lf	1E-37 to 1E+37 along with six digits of the precisions here	64
long double	%Lf	1E-37 to 1E+37 along with six digits of the precisions here	80

4.1.6 Control Statements in C

In C language, the control of the program flows from a given instruction to another. This type of control flow that occurs from any given command to another is known as the sequential control flow. Now, in any C program, a programmer might want to repeat some sets of instructions or even skip the instructions when they are writing logic. Declarations in C, also known as control declarations or decision-making, help them in making such decisions.

Conditional statements are used in the C programming language for making certain decisions on the basis of the available conditions. These conditional statements get executed sequentially in case no condition is present around the statements.

Control Statements Types Used in C Language:

The C language provides support for the following set of statements in its program:

1. If Statements
2. Switch Statement
3. Conditional Operator Statement
4. Goto Statement
5. Loop Statements

1. The If Statements

This type of statement would enable a programmer to choose various instruction sets on the basis of the available condition. The instruction sets will only get executed when the evaluation of the condition turns out to be true. In case the evaluation of the condition is false, there will be an execution of a different instruction set. These are also known as decision control statements. These are of the following types:

- Simple else or Null else
- Else if ladder
- Nested if
- If… else

1.1 The If… Else Statement

When we use the if… else statement, there occurs an execution of two different types of statements in a program. First, if the available condition in the program is true, then there will be an execution of the first statement. The execution of the second condition will only occur if the condition available to us is false.

The syntax for this statement is as follows:

```
If (condition 1)
{
Statement 1 (s1);
}
else
{
```

Statement 2 (s2)

}

1.2 The Nested If Statement

In this case, the condition available in the next if statement (the second statement) will only get evaluated if the evaluation of the condition available in the first statement turns out to be true. This occurs throughout the program that has a nested statement.

The syntax for this statement is as follows:

```
If (condition 1)
{
If (condition 2)
{
Statement 1 (s1);
}
Else
{
Statement 2 (s2)
}
}
```

1.3 The Else If Ladder

In this statement, the execution of an array of instructions occurs only when the available condition is correct. The verification of the next condition occurs when this first condition is incorrect. In case all of the specifications fail even after the verification, then there will be an execution of the default block statements.

The syntax for this statement is as follows:

```
If (condition 1)
{
Statement 1 (s1);
}
Else if (condition 2)
{
Statement 2 (s2);
}
else if (condition 3)
{
```

Statement 3 (s3)

}

Else

{

Statement 4 (s4)

}

Statement (s);

1.4 The Simple Else or Null Else

This condition occurs when a programmer can skip or execute a set of various instructions on the basis of the condition value. We select a one-way, simple statement. When the available condition gets evaluated as true, then a set of various statements will be carried out. In case the condition is false, then the control here will proceed ahead in the program with the declaration mentioned below, after the program's if declaration.

The syntax for this statement is as follows:

If (condition1)

{

Statement 1 (s1);

}

Statement 2 (s2);

2. The Switch Statements

The C language offers its users with a selection statement in various ways in case a program becomes difficult to read with an increased number of conditions. A switch statement is a multi-way type of selection statement that would resolve this issue. The switch declaration comes into play when more than three alternatives (conditions) exist in a program. This command then switches between all the available blocks on the basis of the expression value. Then, each block has a corresponding value with it.

The syntax for this statement is as follows:

Switch (expression_A)

{

Label case_A:

Statement A (sA);

Break;

Label case_B:

Statement B (sB);

Break;

Label case_C;

Statement C (sC);

Break;

….

Label case_Z:

Statement Z (sZ);

Break;

Default:

Statement_1 (s1);

Break;

}

3. The Conditional Operator Statements

The C language also comes with a very unusual operator for its programmers – the conditional operator.

The syntax of the conditional operator statements is as follows:

(condition 1)? expression_1: expression_2

Here, the execution of the expression_1 will only occur when the given condition is valid. In case this statement is incorrect, then the execution of the expression_2 will occur.

4. The Goto Statement

The Goto statement is especially known in the case of jumping control statements. We mainly use the goto statement when we want to transfer a program's control from any one block to another. Also, we use the goto keyword for the declaration of the goto statement.

The syntax of the goto statement is as follows:

goto name_of_label;

name_of_label;

5. The Loop Statements

A programmer in C might want to repeat any set of instructions or certain statements in the program to meet the necessary requirements. In such instances, it becomes difficult to rewrite and repeat everything. And that is exactly where we would like to create loops using the looping declarations. Loop control statements help in such types of situations in C. We have the following types of loops in C:

- Do While Loop
- While Loop
- For Loop

4.1.7 Operators in C

The operators are types of symbols that inform a compiler for performing some specific logical or mathematical functions. The operators serve as the foundations of the programming languages. Thus, the overall functionalities of the C programming language remain incomplete if we do not use operators. In simpler words, the operators are symbols that help users perform specific operations and computations- both logical and mathematical on the operands. Thus, the operators operate on the available operands in a program.

Use of Operators in C

The operators basically serve as symbols that operate on any value or variable. We use it for performing various operations- such as logical, arithmetic, relational, and many more.

The programming languages like C come with some built-in functions that are rich in nature. The use of these operators is vast. These operators act as very powerful and useful features of all the programming languages, and the functionality of these languages is pretty much useless without these. These make it very easy for programmers to write the code very easily and efficiently.

Types of Operators in C

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Assignment Operators
- Bitwise Operators
- Misc Operators

1. Relational Operators

Function of Operator	Operator	Example
To check if two operands are equal to each other.	==	5 == 3 would return to 0 and 4 == 4 would return to 1
To check if two operands are unequal to each other.	!=	8 != 5 would return to 1 6 != 6 would return to 0
To check if the operand value on the left is comparatively greater than that of the operand value on the right.	>	5 > 8 would return to 0 9 > 7 would return to 1

To check if the operand value on the left is comparatively smaller than that of the operand value on the right.	<	2 < 0 would return to 0 4 < 8 would return to 1
To check if the operand value on the left is equal to or comparatively greater than that of the operand value on the right.	>=	1 >= 5 would return to be 0 4 >= 2 would return to be 1 9 >= 9 would return to be 1
To check if the operand value on the left is equal to or comparatively smaller than that of the operand value on the right.	<=	3 <= 6 would return to be 1 7 <= 3 would return to be 0 2 <= 2 would return to be 1

Note – Here, we get 1 as output when the condition is true, and 0 as output when the condition is false.

2. Arithmetic Operators

The arithmetic operators in C language help a user perform the mathematical operations as well as the arithmetic operations in a program, such as subtraction (-), addition (+), division (/), multiplication (*), the remainder of division (%), decrement (–), increment (++).

The arithmetic operators are of two major types:

- **Binary Operators –** It works using two of the operators, such as -, +, /, *.
- **Unary Operators In C –** It works by making use of just a single operand (value), such as — and ++.

2.1 Binary Operators

Here is a table that states all the binary arithmetic operators available in the C language, along with their individual functions. If P = 50 and Q = 25, then:

Function of Operator	Operator	Example
To subtract the value of the second operator from the first one.	–	P – Q = 25
To add the value of the second operator with the first one.	+	P + Q = 75
To divide the value of the numerator with the value of the denominator.	/	P / Q = 2
To multiply the value of the second operator with the first one.	*	P * Q = 1250

2.2 Unary Operators

These are increment and decrement operators. If P = 50 and Q = 25, then:

Function of Operator	Operator	Example
To increase the value of the integer/ operator by 1.	Decrement Operator	P — = 49 and Q — = 24
To decrease the value of the integer/ operator by 1.	Increment Operator	P ++ = 51 and Q ++ = 26

Prefix and Postfix Increment/ Decrement Operators

Type of Operator	Sample Operator Expression	Description/ Explanation
Prefix Increment Operator	++p	p increases by a value of 1, then the program uses the value of p.
Postfix Increment Operator	p++	The program uses the current value of p and increases the value of p by 1.
Prefix Decrement Operator	–p	p decreases by a value of 1, then the program uses the value of p.
Postfix Decrement Operator	p–	The program uses the current value of p and decreases the value of p by 1.

3. Logical Operators

The logical operators in c help a user determine if the results would be true or false. Here is a table that states all the logical operators available in the C language, along with their individual functions. If P = 1 and Q = 0, then:

Name of Operator	Operator	Description of Operator	Example
Logical NOT	!	We use this operator for reversing the available logical state of the operand. In case the condition turns out to be true, then this operator will make the value false.	!(P && Q) turns out to be true.
Logical OR	||	The given condition turns out to be true if any of the operands happen to be non-zero.	(P || Q) turns out to be true.

Logical AND	&&	The given condition turns out to be true if only both the operands happen to be non-zero.	(P && Q) turns out to be false.

4. Assignment Operators

Function of Operator	Operator	Example
To assign the values from the right operand to the left operand.	=	p = q
To add the value of the right operand with the value of the left operand. After that, assign the resultant value to the left one.	+=	p += q is similar to that of p = p=q
To subtract the value of the right operand from the value of the left operand. After that, assign the resultant value to the left one.	-=	p -= q is similar to that of p = p-q
To multiply the value of the right operand with the value of the left operand. After that, assign the resultant value to the left one.	*=	p *= q is similar to that of p = p*q
To divide the value of the right operand with the value of the left operand. After that, assign the resultant value to the left one.	/=	p /= q is similar to that of p = p/q
To calculate the modulus of the values of the right operand and the left operand. After that, assign the resultant value to the left one.	%=	p %= q is similar to that of p = p%q

5. Bitwise Operators

The Bitwise operators basically work on the bits, and we perform bit-by-bit operations using them. Here is the truth table for the ^, &, and | here:

a	**b**	**a & b**	**a | b**	**a ^ b**
1	1	1	1	0
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1

6. Misc Operators

Description of Operator	Operator	Example

To return the actual size of any given variable.	Size of()	If z is an integer, then the size of(z) will return to be 4.
To return the address of any given variable.	&	For instance, &z; would return the actual address for any given variable.
Pointer to any variable.	*	*z;
Conditional Expression.	? :	When the Condition is true ? then the value X : In any other case, the value Y

Precedence of Operators in C

Type of Operator	**Associativity**	**Category**
() [] -> . ++ – –	Left to right	Postfix
– + ! ~ – – ++ (type)* & sizeof	Right to left	The Unary Operator
/ * %	Left to right	The Multiplicative Operator
– +	Left to right	The Additive Operator
>> <<	Left to right	The Shift Operator
< > >= <=	Left to right	The Relational Operator
!= ==	Left to right	The Equality Operator
&	Left to right	Bitwise AND
^	Left to right	Bitwise XOR
|	Left to right	Bitwise OR
&&	Left to right	Logical AND
||	Left to right	Logical OR
?:	Right to left	Conditional
= -= += /= *= %=>>= &= <<= |= ^=	Right to left	Assignment
,	Left to right	Comma

4.1.8 Pointers in C

The pointers in C language refer to the variables that hold the addresses of different variables of similar data types. We use pointers to access the memory of the said variable and then manipulate their addresses in a program. The pointers are very distinctive features in C- it provides the language with flexibility and power.

What are Pointers in C?

The pointers perform the function of storing the addresses of other variables in the program. These variables could be of any type- char, int, function, array, or other pointers. The pointer sizes depend on their architecture. But, keep in mind that the size of a pointer in the 32-bit architecture is 2 bytes.

Let us look at an example where we define a pointer storing an integer's address in a program.

int x = 10;

int* p = &x;

Here, the variable p is of pointer type, and it is pointing towards the address of the x variable, which is of the integer type.

How Do We Use Pointers in C?

Consider that we are declaring a variable "a" of int type, and it will store a value of zero.

int a = 0

Now, a is equal to zero.

Declaration of a Pointer

Just like the variables, we also have to declare the pointers in C before we use them in any program. We can name these pointers anything that we like, as long as they abide by the naming rules of the C language. Normally, the declaration of a pointer would take a form like this: data_type * name_of_pointer_variable;

Note that here,

- The data_type refers to this pointer's base type in the variable of C. It indicates which type of variable is the pointer pointing to in the code.
- The asterisk (*) is used to declare a pointer. It is an indirection operator, and it is the same asterisk that we use in multiplication.

We can declare pointers in the C language with the use of the asterisk symbol (*). It is also called the indirection pointer, as we use it for dereferencing any pointer. Here is how we use it:

int *q; // a pointer q for int data type

char *x; // a pointer x for char data type

The Initialization of a Pointer

Once we declare a pointer, we then initialise it just like the standard variables using an address of the variable. In case we don't initialise the pointers in any C program and start using it directly, the results can be pretty unpredictable and potentially disastrous.

We use the & (ampersand) operator to get the variable's address in the program. We place the & just before that variable's name whose address we require. Here is the syntax that we use for the initialisation of a pointer,

Syntax of Pointer Initialization

pointer = &variable;

As we can assess in the figure shown above, the pointer variable is storing the digit variable's address, named fff4. This digit variable's value is equal to 100. But aaa3 is the pointer variable's address (variable r).

Here, we can print the *r* pointer variable's value using the indirection pointer (*).

Use of Pointers in C

Here is a summary of how we use the following operators for the pointers in any program:

Operator	Name	Uses and Meaning of Operator
*	Asterisk	Declares a pointer in a program. Returns the referenced variable's value.
&	Ampersand	Returns a variable's address

The Pointer to an Array

Here is an example to illustrate the same,

int arr [20];

int *q [20] = &arr; // The q variable of the pointer type is pointing towards the integer array's address or the address of arr.

The Pointer to a Function

Here is an example to illustrate the same,

void display (int);

void(*q)(int) = &show; // The q pointer is pointing towards the function's address in the program

The Pointer to a Structure

Here is an example to illustrate the same,

struct str {

int x;

float y;

}ref;

struct str *q = &ref;

Types of Pointers

There are various types of pointers that we can use in the C language. Let us take a look at the most important ones.

1. The Null Pointer

The null pointer has a value of 0. To create a null pointer in C, we assign the pointer with a null value during its declaration in the program. This type of method is especially useful when the pointer has no address assigned to it.

2. The Void Pointer

The void pointer is also known as the generic pointer in the C language. This pointer has no standard data type, and we create it with the use of the keyword *void*. The void pointer is generally used for the storage of any variable's address

3. The Wild Pointer

We can call a pointer a wild pointer if we haven't initialized it with anything. Such wild pointers are not very efficient in a program, as they may ultimately point towards some memory location that is unknown to us. It will ultimately lead to some problems in the program, and then, it will crash. Thus, you must be very careful when using wild pointers in a program.

- Near pointer
- Complex pointer
- Huge pointer
- Far pointer
- Dangling pointer

4. Accessing Pointers- Indirectly and Directly

There are basically two ways in which a program can access a variable content and then manipulate it:

- **Direct Access**: In this case, we use the variable name in the program directly.
- **Indirect Access**: In this case, we use some pointers to the variables in the program.

Applications of Pointers in C

There are various uses of the pointers in C language. Let us look at some of them:

1. Dynamic allocation of memory: We can allocate the memory dynamically in C language when we use the calloc() and malloc() functions. The pointers are primarily used in such cases.

2. Structures, Functions, and Arrays: We generally use the pointers in the C language in the cases of structures, functions, and arrays. Here, the pointers help in reducing the code and improving the program's performance.

4.1.9 File Handling in C

The process of file handling refers to how we store the available data or info in a file with the help of a program. The C language stores all the data available in a program into a file with the help of file handling in C. This data can be fetched/extracted from these files to work again in any program.

What is File Handling in C?

File handling refers to the method of storing data in the C program in the form of an output or input that might have been generated while running a C program in a data file, i.e., a binary file or a text file for future analysis and reference in that very program.

What is a File in C?

A file refers to a source in which a program stores the information/data in the form of bytes of sequence on a disk (permanently). The content available on a file isn't volatile like the compiler memory in C. But the program can perform various operations, such as creating, opening, reading a file, or even manipulating the data present inside the file. This process is known as file handling in C.

Why Do We Need File Handling in C?

There are times when the output generated out of a program after its compilation and running do not serve our intended purpose. In such cases, we might want to check the program's output various times. Now, compiling and running the very same program multiple times becomes a tedious task for any programmer. It is exactly where file handling becomes useful.

- **Reusability:** File handling allows us to preserve the information/data generated after we run the program.
- **Saves Time:** Some programs might require a large amount of input from their users. In such cases, file handling allows you to easily access a part of a code using individual commands.
- **Commendable storage capacity:** When storing data in files, you can leave behind the worry of storing all the info in bulk in any program.
- **Portability:** The contents available in any file can be transferred to another one without any data loss in the computer system. This saves a lot of effort and minimises the risk of flawed coding.

Types of Files in a C Program

When referring to file handling, we refer to files in the form of data files. Now, these data files are available in 2 distinct forms in the C language, namely:

- Text Files
- Binary Files

1. Text Files

The text files are the most basic/simplest types of files that a user can create in a C program. We create the text files using an extension *.txt* with the help of a simple text editor. In general, we can use notepads for the creation of .txt files. These files store info internally in ASCII character format, but when we open these files, the content/text opens in a human-readable form.

Text files are, thus, very easy to access as well as use. But there's one major disadvantage; it lacks security. Since a .txt file can be accessed easily, information isn't very secure in it. Added to this, text files consume a very large space in storage.

To solve these problems, we have a different type of file in C programs, known as binary files.

2. Binary Files

The binary files store info and data in the binary format of 0's and 1's (the binary number system). Thus, the files occupy comparatively lesser space in the storage. In simpler words, the binary files store data and info the same way a computer holds the info in its memory. Thus, it can be accessed very easily as compared to a text file.

The binary files are created with the extension .bin in a program, and it overcomes the drawback of the text files in a program since humans can't read it; only machines can. Thus, the information becomes much more secure. Thus, binary files are safest in terms of storing data files in a C program.

Operators/Functions that We Use for File Handling in C

We can use a variety of functions in order to open a file, read it, write more data, create a new file, close or delete a file, search for a file, etc. These are known as file handling operators in C.

Here's a list of functions that allow you to do so:

Description of Function	Function in Use
used to open an existing file or a new file	fopen()
writing data into an available file	fprintf()
reading the data available in a file	fscanf()
writing any character into the program file	fputc()
reading the character from an available file	fgetc()
used to close the program file	fclose()
used to set the file pointer to the intended file position	fseek()
writing an integer into an available file	fputw()
used to read an integer from the given file	fgetw()
used for reading the current position of a file	ftell()
sets an intended file pointer to the file's beginning itself	rewind()

Note: It is important to know that we must declare a file-type pointer when we are working with various files in a program. This helps establish direct communication between a program and the files.

Here is how you can do it:

FILE *fpointer;

Out of all the operations/functions mentioned above, let us discuss some of the basic operations that we perform in the C language.

Operations Done in File Handling

The process of file handling enables a user to update, create, open, read, write, and ultimately delete the file/content in the file that exists on the C program's local file system. Here are the primary operations that you can perform on a file in a C program:

- Opening a file that already exists
- Creating a new file
- Reading content/ data from the existing file
- Writing more data into the file
- Deleting the data in the file or the file altogether

Opening a File in the Program – to create and edit data

We open a file with the help of the fopen() function that is defined in the header file- stdio.h.

Here is the syntax that we follow when opening a file:

ptr = fopen ("openfile" , "openingmode");

Let us take a look at an example for the same,

fopen ("E:\\myprogram\\recentprogram.txt" , "w");

fopen ("E:\\myprogram\\previousprogram.bin" , "rb");

Let us take a look at a few more opening modes used in the C programs:

Opening Modes of C in Standard I/O of a Program		
Mode in Program	**Meaning of Mode**	**When the file doesn't exist**
r	Open a file for reading the content.	In case the file doesn't exist in the location, then fopen() will return NULL.
rb	Open a file for reading the content in binary mode.	In case the file doesn't exist in the location, then fopen() will return NULL.
w	Open a file for writing the content.	In case the file exists, its contents are overwritten.

		In case the file doesn't exist in the location, then it will create a new file.
wb	Open a file for writing the content in binary mode.	In case the file exists, then its contents will get overwritten. In case the file doesn't exist in the location, then it will create a new file.
a	Open a file for appending the content. Meaning, the data of the program is added to the file's end in a program.	In case the file doesn't exist in the location, then it will create a new file.
ab	Open a file for appending the content in binary mode. Meaning, the data of the program is added to the file's end in a program in a binary mode.	In case the file doesn't exist in the location, then it will create a new file.
r+	Open a file for both writing and reading the content.	In case the file doesn't exist in the location, then fopen() will return NULL.
rb+	Open a file for both writing and reading the content in binary mode.	In case the file doesn't exist in the location, then fopen() will return NULL.
w+	Open a file for both writing and reading.	In case the file exists, its contents are overwritten. In case the file doesn't exist in the location, then it will create a new file.
wb+	Open a file for both writing and reading the content in binary mode.	In case the file exists, its contents are overwritten. In case the file doesn't exist in the location, then it will create a new file.
a+	Open a file for both appending and reading the content.	In case the file doesn't exist in the location, then it will create a new file.
ab+	Open a file for both appending and reading the content in binary mode.	In case the file doesn't exist in the location, then it will create a new file.

How do we close a file?

Once we write/read a file in a program, we need to close it (for both binary and text files). To close a file, we utilise the fclose() function in a program.

Here is how the program would look like:

fclose(fptr);

In this case, the fptr refers to the file pointer that is associated with that file that needs to be closed in a program.

How do we read and write the data to the text file?

We utilise the fscanf() and fprintf() to write and read the data to the text file. These are basically the file versions of the scanf() and printf(). But there is a major difference, i.e., both fscanf() and fprintf() expect a pointer pointing towards the structure FILE in the program.

// if you are using Linux or MacOS, then you must use appropriate locations

fptr = fopen ("C:\\currentprogram.txt","w");

if(fptr == NULL)

{

printf("File type invalid!");

exit(1);

}

printf("Please enter the val: ");

scanf("%d",&val);

fprintf(fptr,"%d",val);

fclose(fptr);

return 0;

}

The program mentioned here would take the number given by then store it in the currentprogram.txt file.

Once we compile this program and run it on the system, we will be able to witness a currentprogram.txt text file created in the C drive of our computer! Also, when we open this file, then we will be able to see what integer we entered as an input while coding.

4.1.10 Use of C and Key Applications

C is one of the oldest and most fundamental programming languages, and it is extensively used all over the world. C is a fast, portable language with a large library. It is a middle-level language with the advantages of both low-level and high-level languages. And it's disheartening to learn that C programming is becoming less popular by the day. C has left an indelible mark on practically every field and is widely used for application development and system development.

Some applications of the C programming language include:

1. Operating System

The C programming language was created with the intention of writing UNIX operating systems. Furthermore, the execution time of programmes written in C is comparable to that of assembly language, making C the most important component in the development of multiple operating systems. It was used to write the Unix kernel, Microsoft Windows utilities and operating system apps, and a large portion of the Android operating system.

2. 3D Movies

Applications written in C and C++ are commonly used to make 3D videos, because they handle a large quantity of data and do many computations per second, these apps must be extremely efficient and quick. The less time it takes for designers and animators to create movie shots, the more money the corporation saves.

3. Intermediate Language

C is occasionally used by implementations of other languages as an intermediate language. This method can be used for portability or convenience, as it eliminates the need for machine-specific code generators by using C as an intermediate language. C includes certain characteristics that aid compilation of generated code, such as line-number preprocessor directives and optional unnecessary commas at the end of initializer lists. However, some of C's flaws have encouraged the creation of additional C-based languages, such as, that are expressly designed for usage as intermediate languages.

4. Play Important Role in Development of New Programming Language

The program written in C is easy and quick to execute. As a consequence, the C programming language has resulted in the creation of many other languages. C++ (also known as C with classes), C#, Python, Java, JavaScript, Perl, PHP, Verilog, D, Limbo, and the Unix C shell are examples of these languages. Every language employs the C programming language to varying degrees. Python, for example, uses C to provide standard libraries.

5. Embedded Systems

The C programming language is the recommended language for creating embedded system drivers and applications. The availability of machine-level hardware APIs, as well as the presence of C compilers, dynamic memory allocation, and deterministic resource consumption, make this language more popular.

6. Portability and Efficiency

C is an assembly language that may be used anywhere. It's as near to the machine as possible while being virtually compatible with existing processor designs. And practically every technology, there is at least one C compiler. Nowadays, because new compilers generate highly optimized binaries, improving their output using hand-written assembly is difficult.

7. Memory Manipulation

C's ability to access arbitrary memory addresses and perform pointer arithmetic is a key feature that makes it ideal for system programming (operating systems and embedded systems).

Computer systems and microcontrollers map their peripherals and I/O pins into memory locations at the hardware/software boundary. To communicate with the outside world, system applications must read and write to those particular memory regions. As a result, the ability of C to manipulate independent memory addresses is critical for system programming.

8. Resources Are Used in a Definite Way

Garbage collection, or even dynamic allocation for certain embedded systems, is a common language feature that system programming cannot rely on. Embedded apps have a certain amount of time and memory. They're frequently used in real-time systems in which a non-deterministic garbage collector call isn't an option. If dynamic allocation is not possible due to a paucity of memory, it is critical to have additional memory management mechanisms in place, such as the ability to place data in custom addresses using C pointers. Languages that rely significantly on dynamic allocation and trash collection would be unsuitable for resource-constrained environments.

4.2 Introduction To Compilers

A Compiler is a software that typically takes a high level language (Like C++ and Java) code as input and converts the input to a lower level language at once. It lists all the errors if the input code does not follow the rules of its language. This process is much faster than interpreter but it becomes difficult to debug all the errors together in a program.

A compiler is a translating program that translates the instructions of high level language to machine level language. A program which is input to the compiler is called a **Source program**. This program is now converted to a machine level language by a compiler is known as the **Object code.**

There are different Compilers :

- **Cross-Compiler –** The compiled program can run on a computer whose CPU or Operating System is different from the one on which the compiler runs.
- **Bootstrap Compiler –** The compiler written in the language that it intends to compile.
- **Decompiler –** The compiler that translates from a low-level language to a higher level one.
- **Transcompiler –** The compiler that translates high level languages.

A compiler can translate only those source programs which have been written in the language for which the compiler is meant. Each high level programming language requires a separate compiler for the conversion.

For Example, a **FORTRAN** compiler is capable of translating into a **FORTRAN** program. A computer system may have more than one compiler to work for more than one high level languages.

Top most Compilers used according to the Computer Languages –

- **C**– Turbo C, Tiny C Compiler, GCC, Clang, Portable C Compiler
- **C++ -**GCC, Clang, Dev C++, Intel C++, Code Block
- **JAVA**– IntelliJ IDEA, Eclipse IDE, NetBeans, BlueJ, JDeveloper
- **Kotlin**– IntelliJ IDEA, Eclipse IDE
- **Python**– CPython, JPython, Wing, Spyder
- **JavaScript**– WebStorm, Atom IDE, Visual Studio Code, Komodo Edit

5. Modules of Proposed System

This proposed system consists of 3 main modules, which are listed below.

1. ADMINISTRATIVE MODULE

Online Voting is a voting system by which any Voter can use his\her voting rights from anywhere in India. Online voting for association contains-:

- Voter's information in database.
- Voter's Names with ID.
- Voter's vote in a database.
- Calculation of total number of votes

Various operational works that are done in the system are:-

- Recording information of the Voter in Voter database.
- Checking of information filled by voter.
- Discard the false information.
- Each information is maintained by admin.
- Delete Illegal Votes
- Ban UserIDs

2. Nominee Candidate Module

The Nominee details will be updated by the admin for the post of board of director and manager. The candidate will submit their own details and the admin maintain all of background details of the particular nominee and uploaded their information in correct procedure. In order to, the user or voter can view the nominee details.

3. USER/VOTER MODULE

The user after their registration or entering the user credentials only can login for voting. The user will view
nominee details can vote. After knowing the nominee details the user can login for voting. They should vote for board of director and the
manager in the association. The count will taken for each voting. After voting the particular person/user cannot logon to vote.

6. System Design

6.1 Data Flow Diagram :

The data flow diagram(DFD) is a graphical tool used for expressing system requirements in a graphical form. The DFD also known as the “bubble chart” as the purpose of clarification system requirements and identification major transformation that will become program in system design. Thus DFD can be stated as the starting point of the design phase that functionality decomposes the requirements specification down to the lowest level of details. The DFD consists of series of bubble joined by lines. The bubble represents data transformation and the lines represents the data flow in the system.

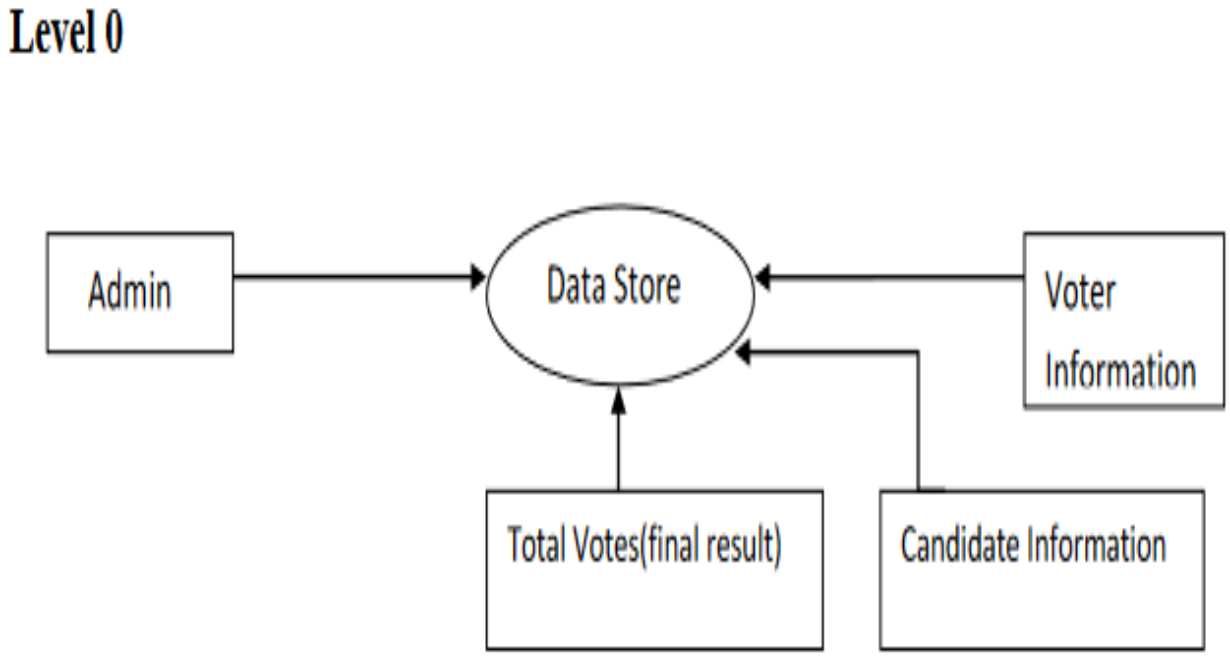
A DFD describes what data flow is does not to construct a Data Flow Diagram, we use

- **Arrow**: An arrow identifies the data flow in motion. It is a pipeline through which information is flow like the rectangle in the flowchart.

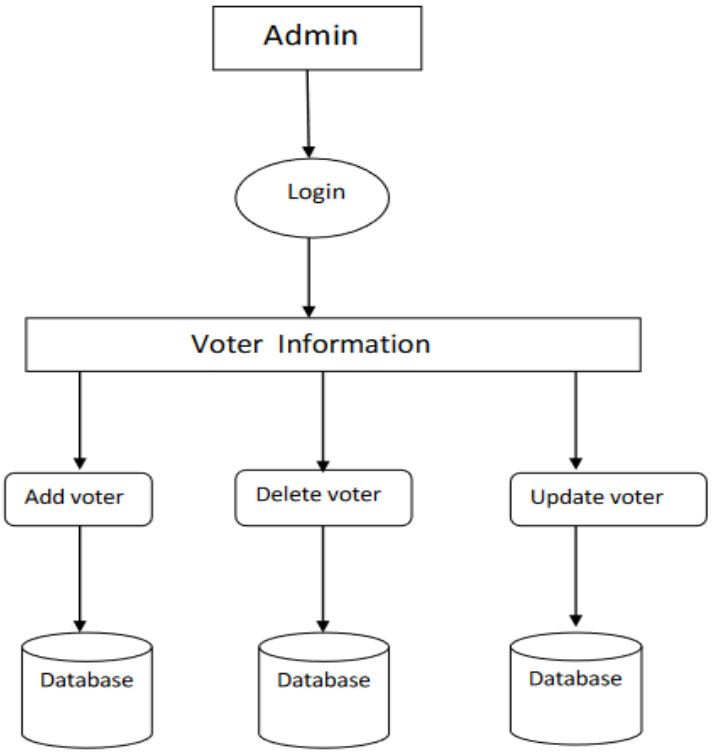
- **Circle**: A circle stands for process that converts data into information.

- **Open End Box**: An open ended box represents a data store, data at rest or a temporary repository of data.

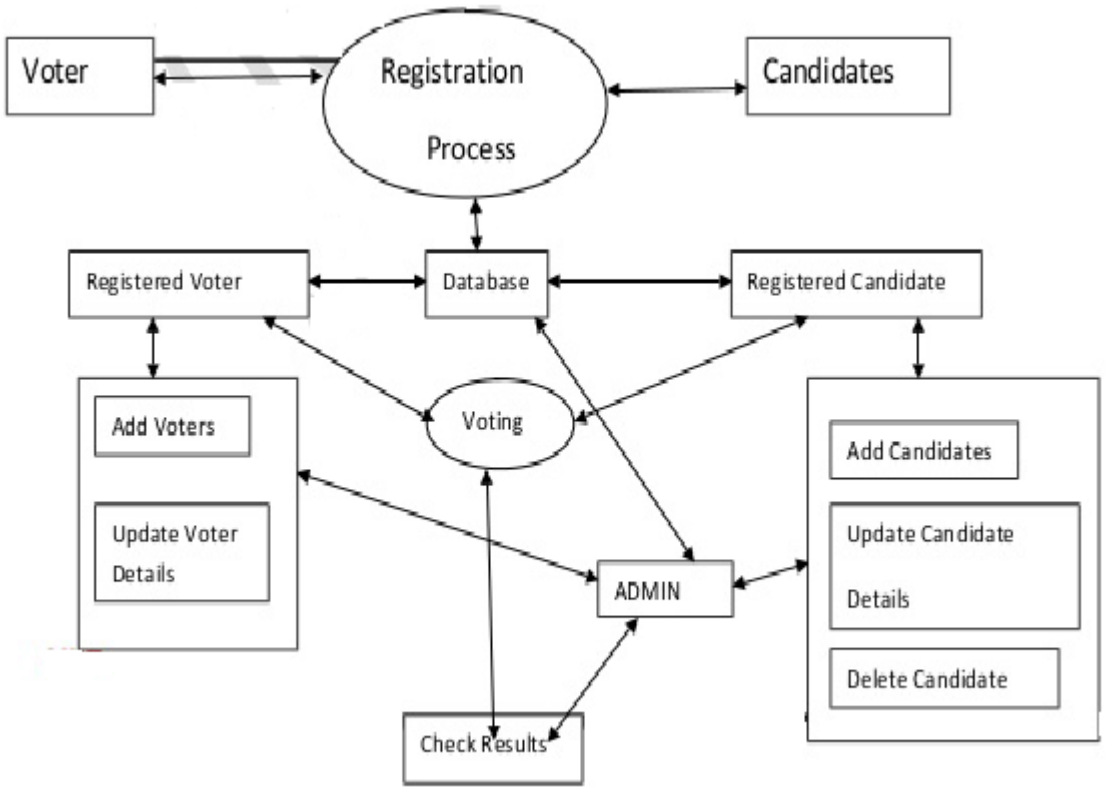
- **Squares**: A square defines a source or destination of system.



Level 1



6.2 E-R Diagram :



7. Testing and Result

7.1 Types Of Testing

7.1.1 System Testing:

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and coding. The user tests the developed system and changes are made according to their needs. The testing phase involves the testing developed system using various kinds of data.

System is the stage of implementation that is aimed at assuring at the system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct, the goal will be successfully achieved. The candidate system is subject to a variety of tests such as recover, security and usability tests. A series of testing is performed for the proposed system before the system is ready for the user acceptance testing.

7.1.2 Component testing:

- Testing of individual program components i.e. the each module is tested
- Usually the responsibility of the component developer (except sometimes for critical systems);
- Tests are derived from the developer's experience.
- Component or unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- Components may be:
- Individual functions or methods within an object;
- Object classes with several attributes and methods; Composite components with defined interfaces used to access their functionality

7.2 Testing Strategies

Following are few of the testing strategies used for the testing purpose:

- Unit testing.
- Validation testing.
- Output testing.
- User acceptance testing.

7.3 Testing Guidelines

Testing guidelines are hints for the testing team to help them choose tests that will reveal defects in the system.

- Choose inputs that force the system to generate all error messages;
- Design inputs that cause buffers to overflow;
- Repeat the same input or input series several times;
- Force invalid outputs to be generated;
- Force computation results to be too large or too small

7.4 Test Case Design

- Involves designing the test cases (inputs and outputs) used to test the system.
- The goal of test case design is to create a set of tests that are effective in validation and defect testing.
- Design approaches:
  - Requirements-based testing;
  - Partition testing;
  - Structural testing.

7.4.1 Requirements based testing:

- A general principle of requirements engineering is that requirements should be testable. Requirements-based testing is a validation testing technique where you consider each requirement and derive a set of tests for that requirement.

7.4.2 Partition testing:

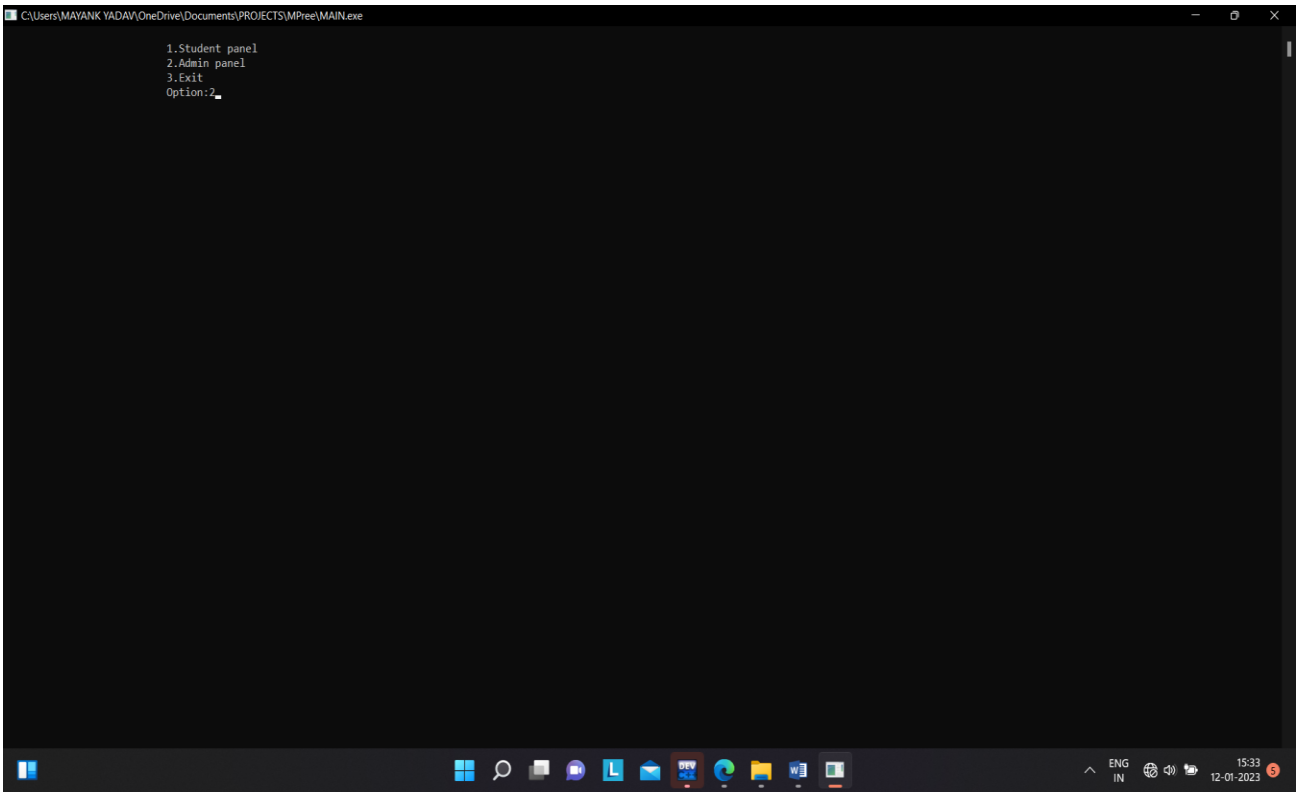
- Input data and output results often fall into different classes where all members of a class are related.
- Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member.
- Test cases should be chosen from each partition.

7.4.3 Structural testing:

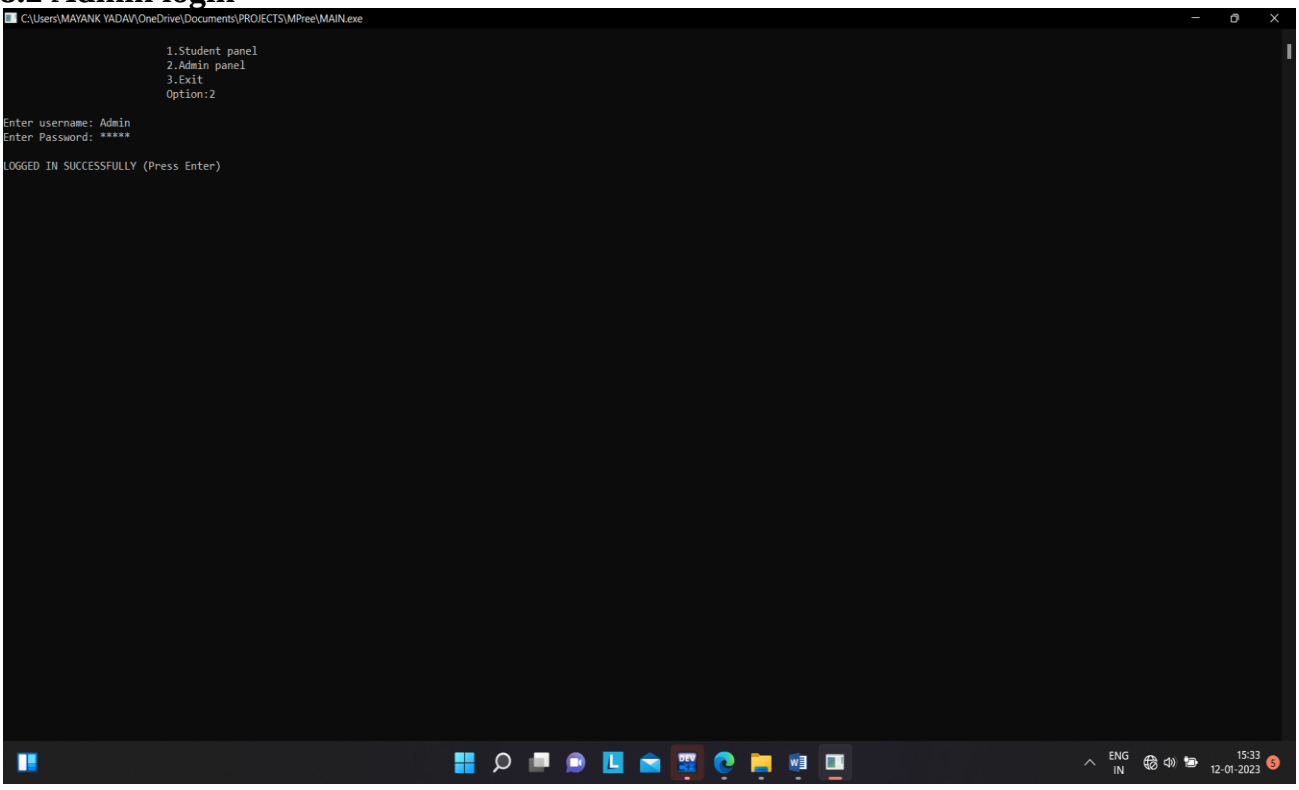
- Sometime called white-box testing.
- Derivation of test cases according to program structure. Knowledge of the program is used to identify additional test cases.
- Objective is to exercise all program statements.

8. Snapshots

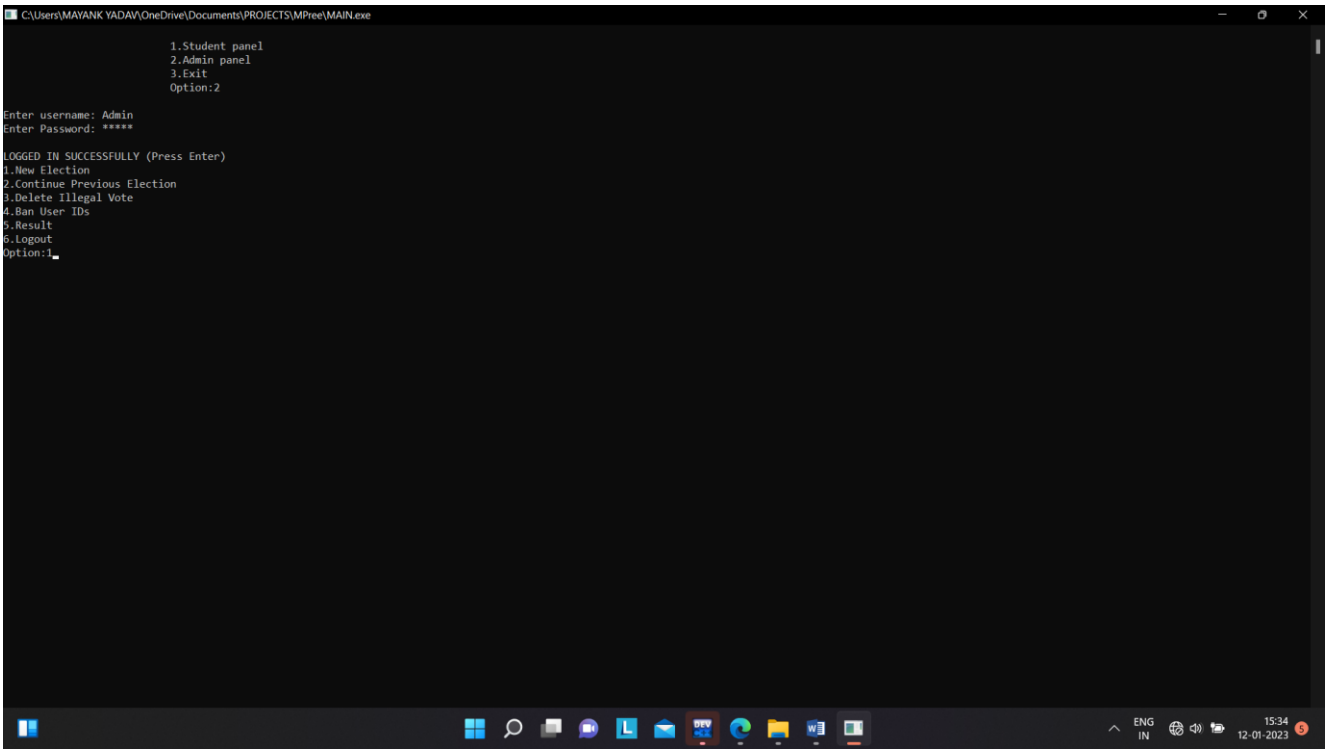
8.1 Home Page



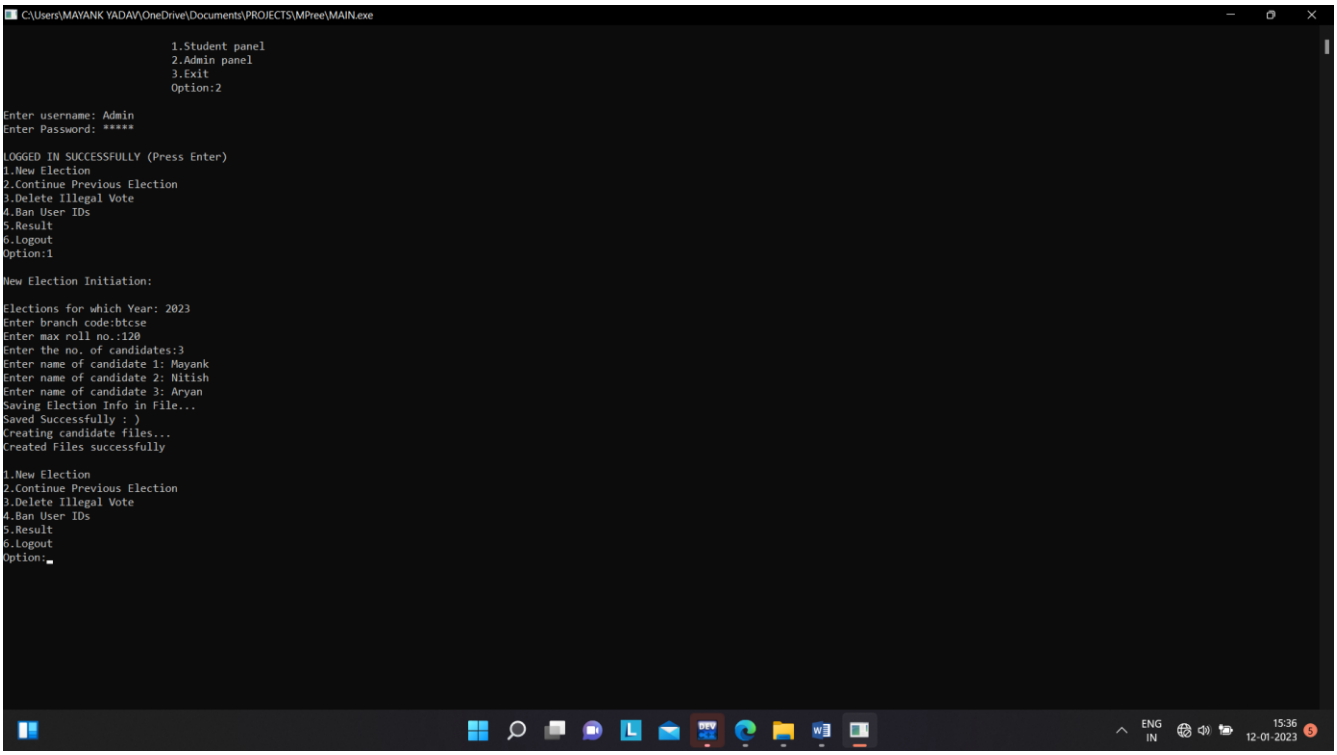
8.2 Admin login



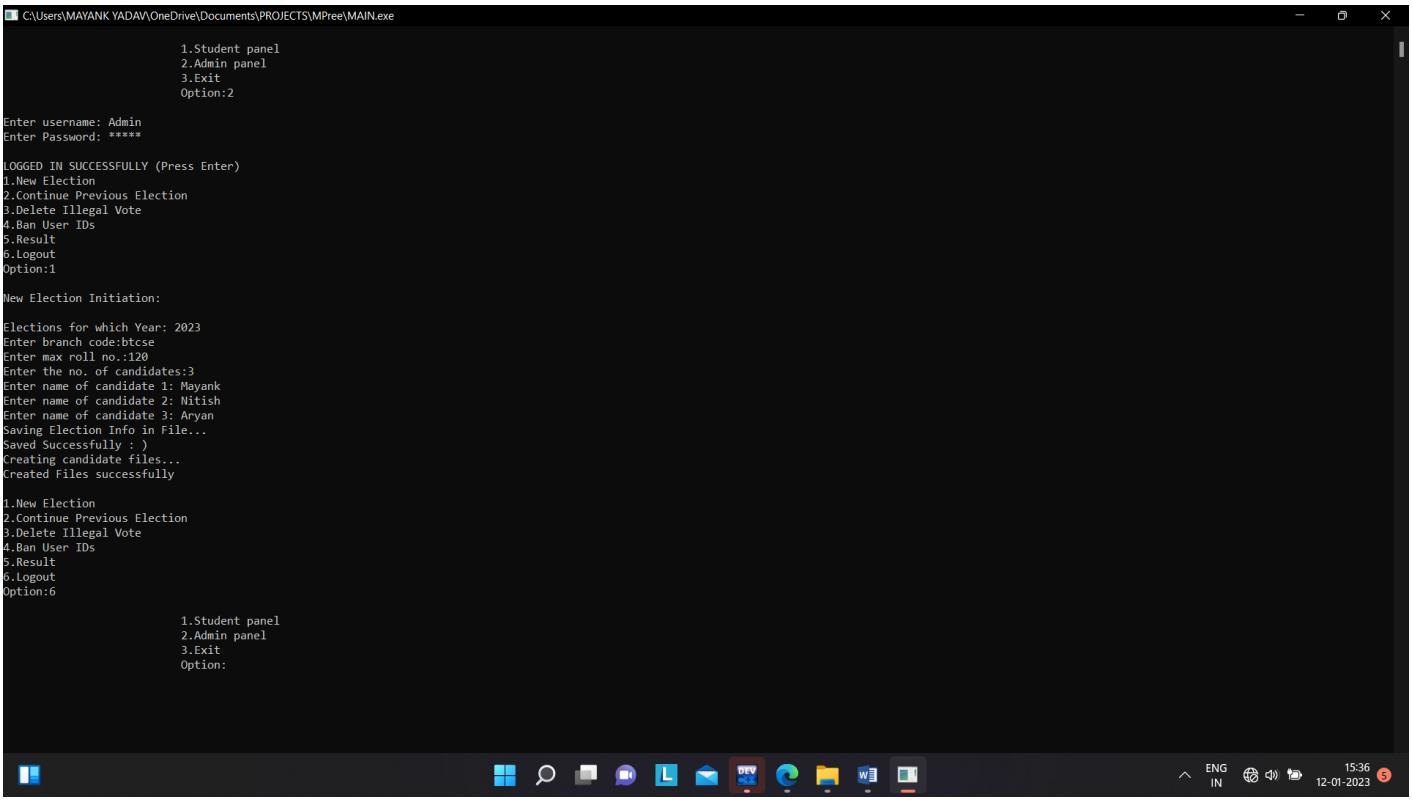
8.3 Admin login page



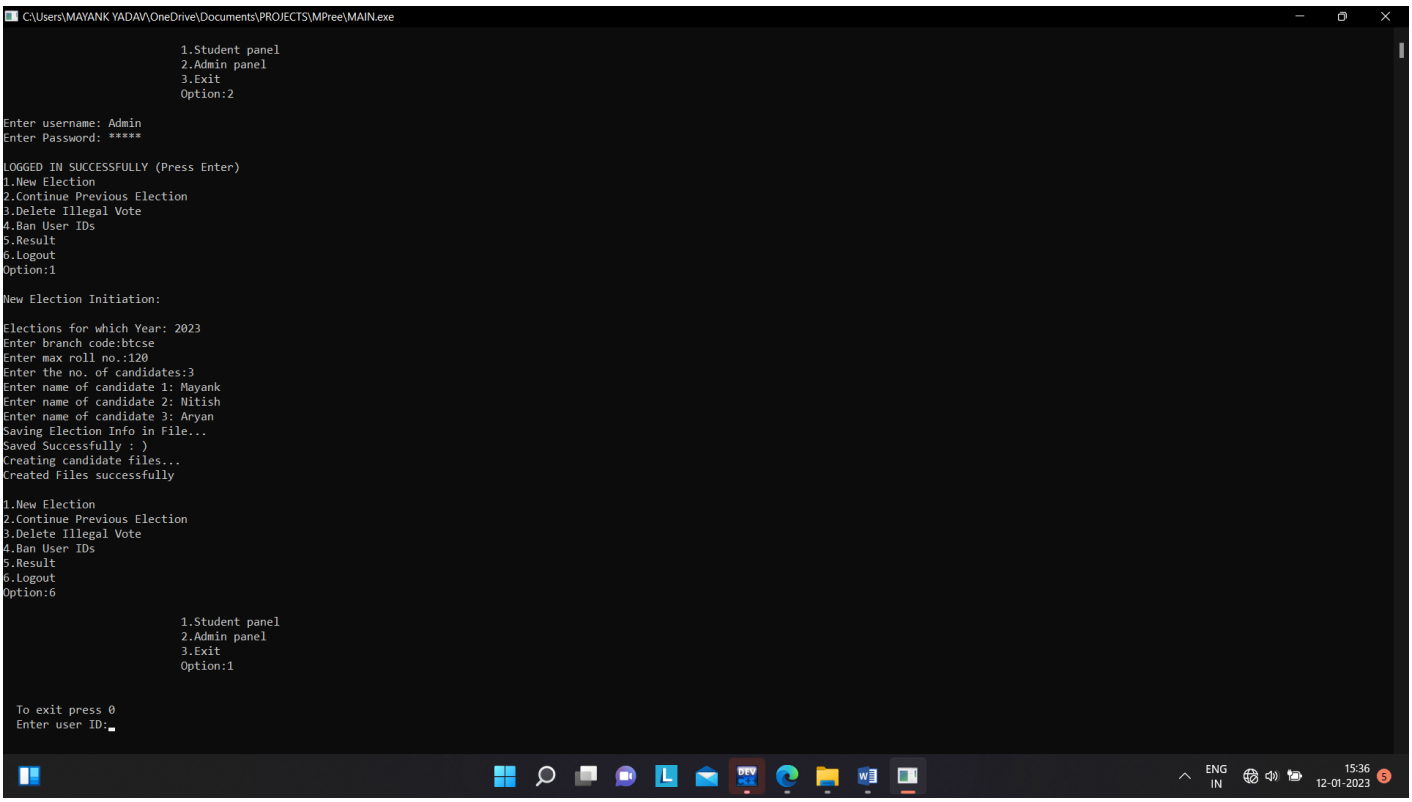
8.4 New Election and Enter Details and Candidate Names



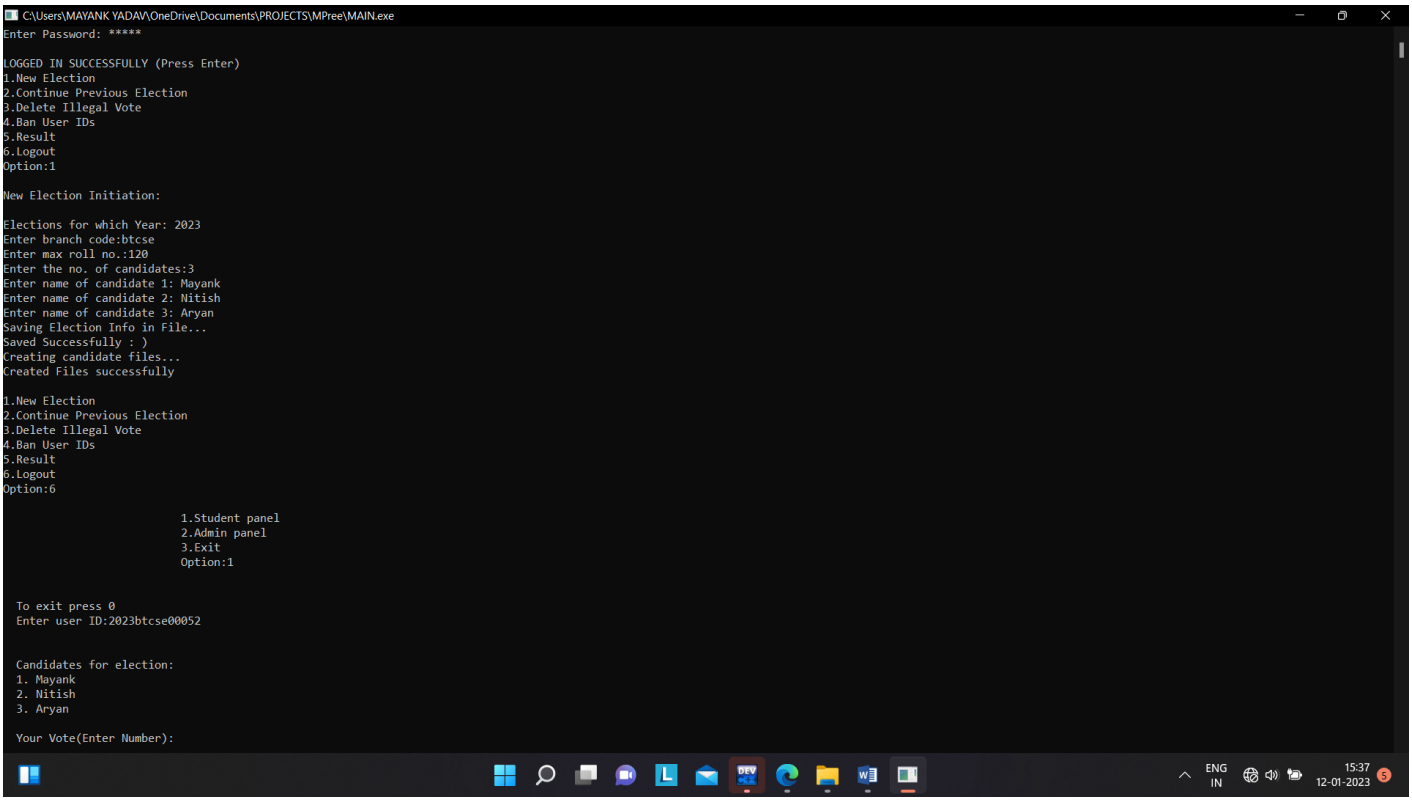
8.5 Logout



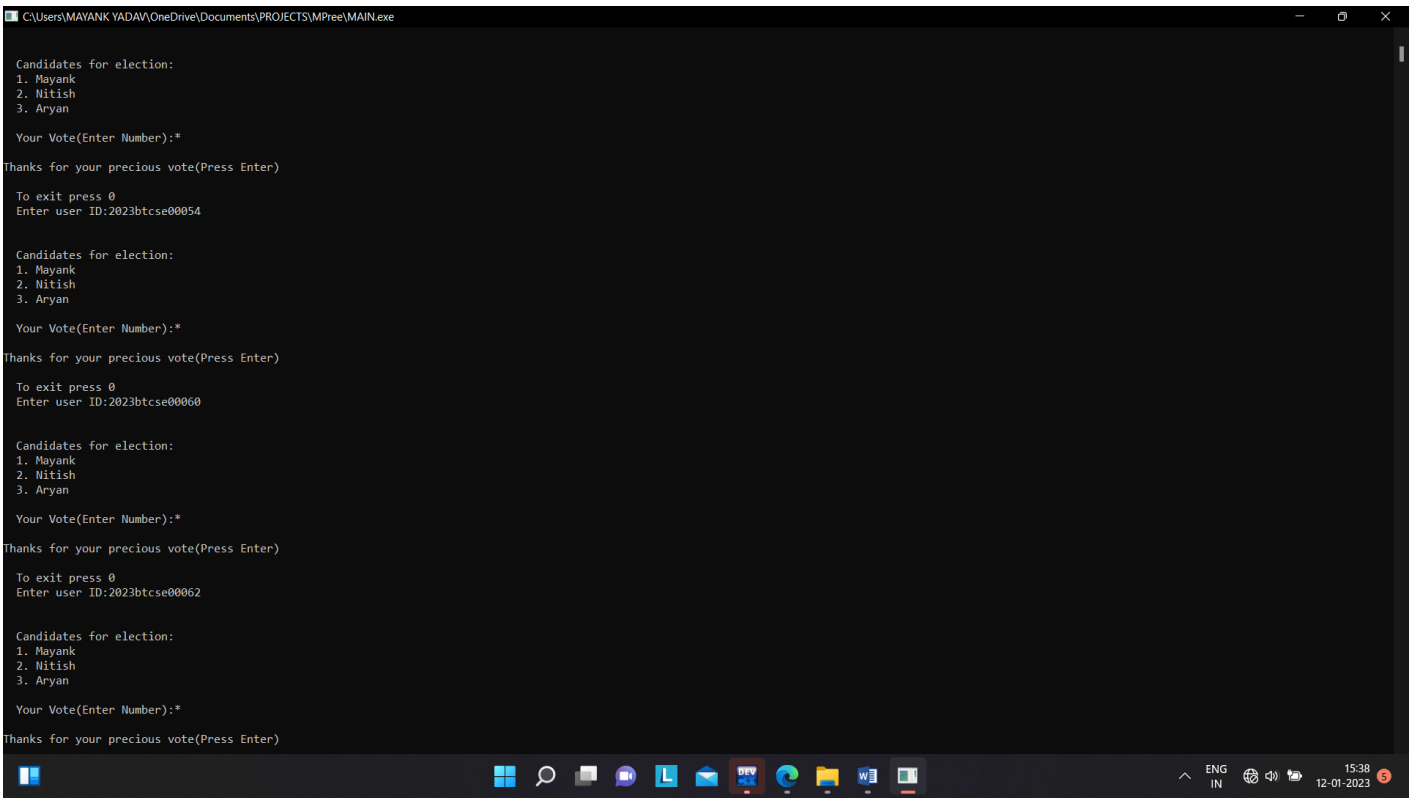
8.6 Enter Student Panel



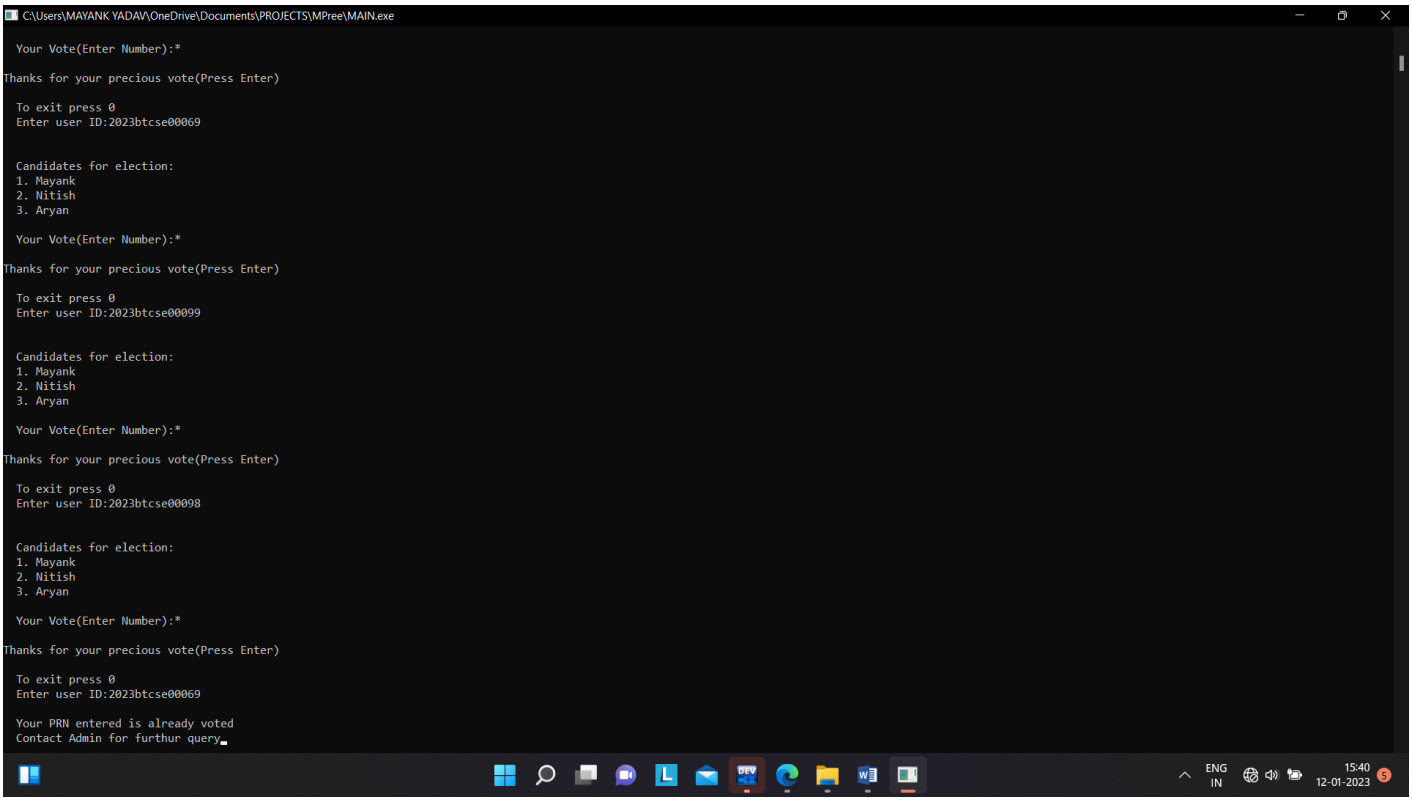
8.7 Enter User ID



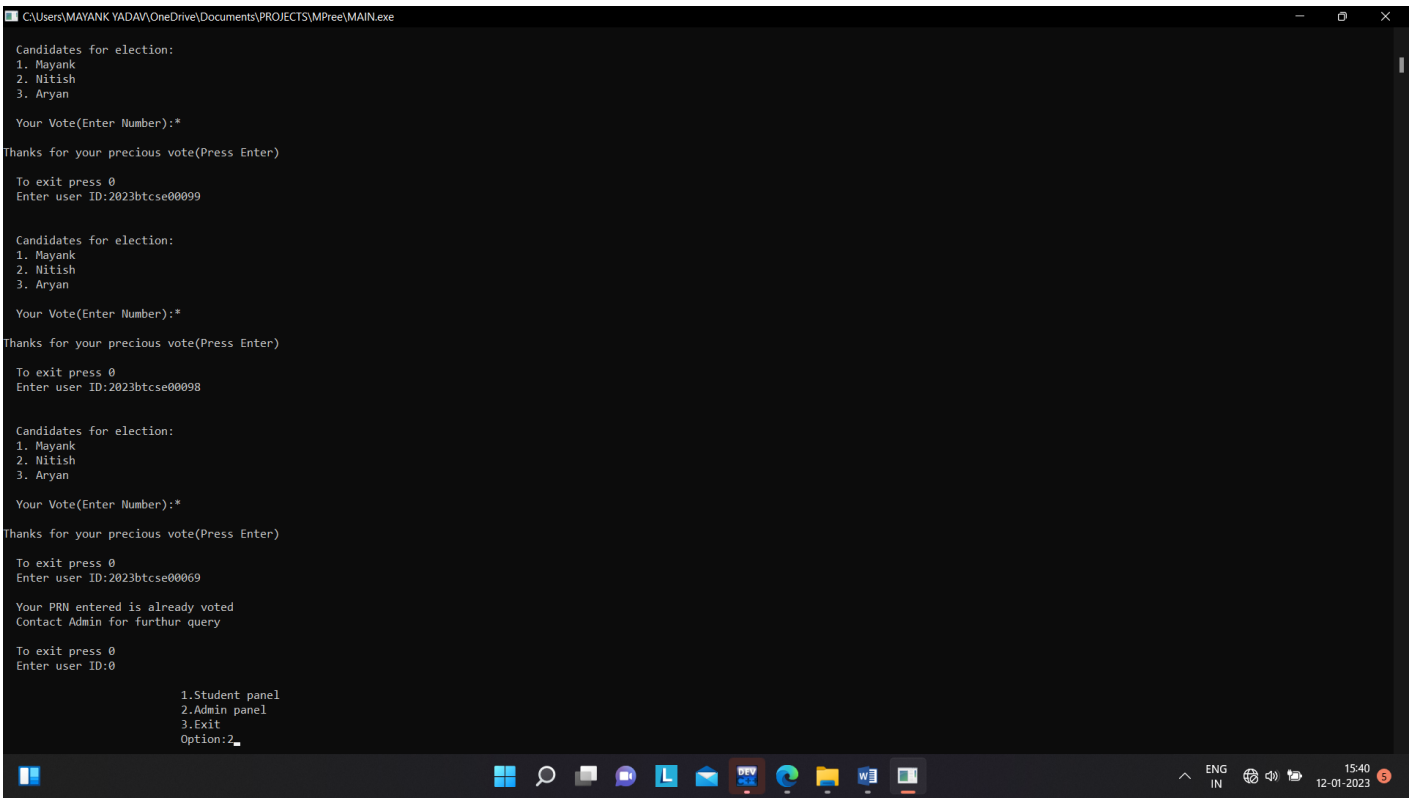
8.8 Voting using user IDs



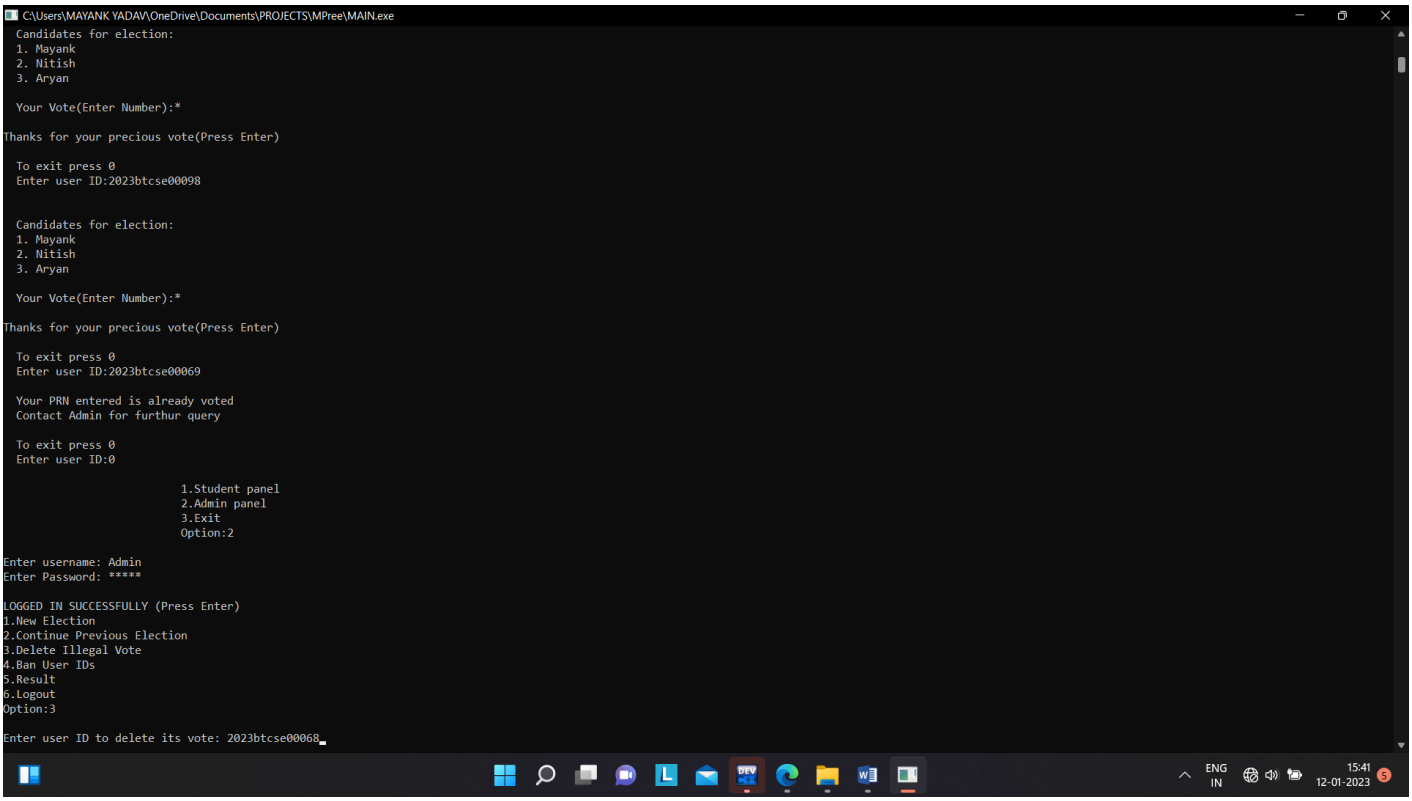
8.9 Already voted



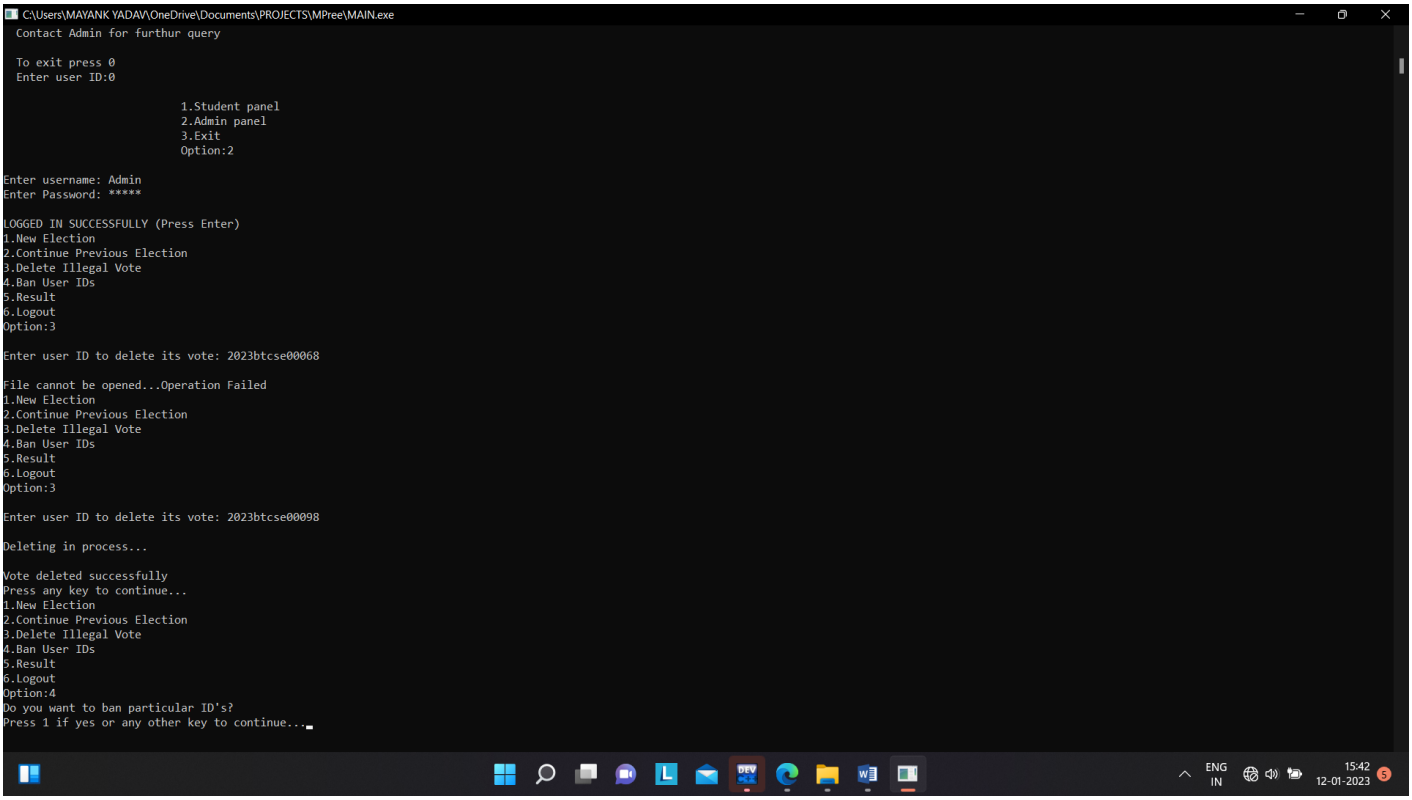
8.10 Exit



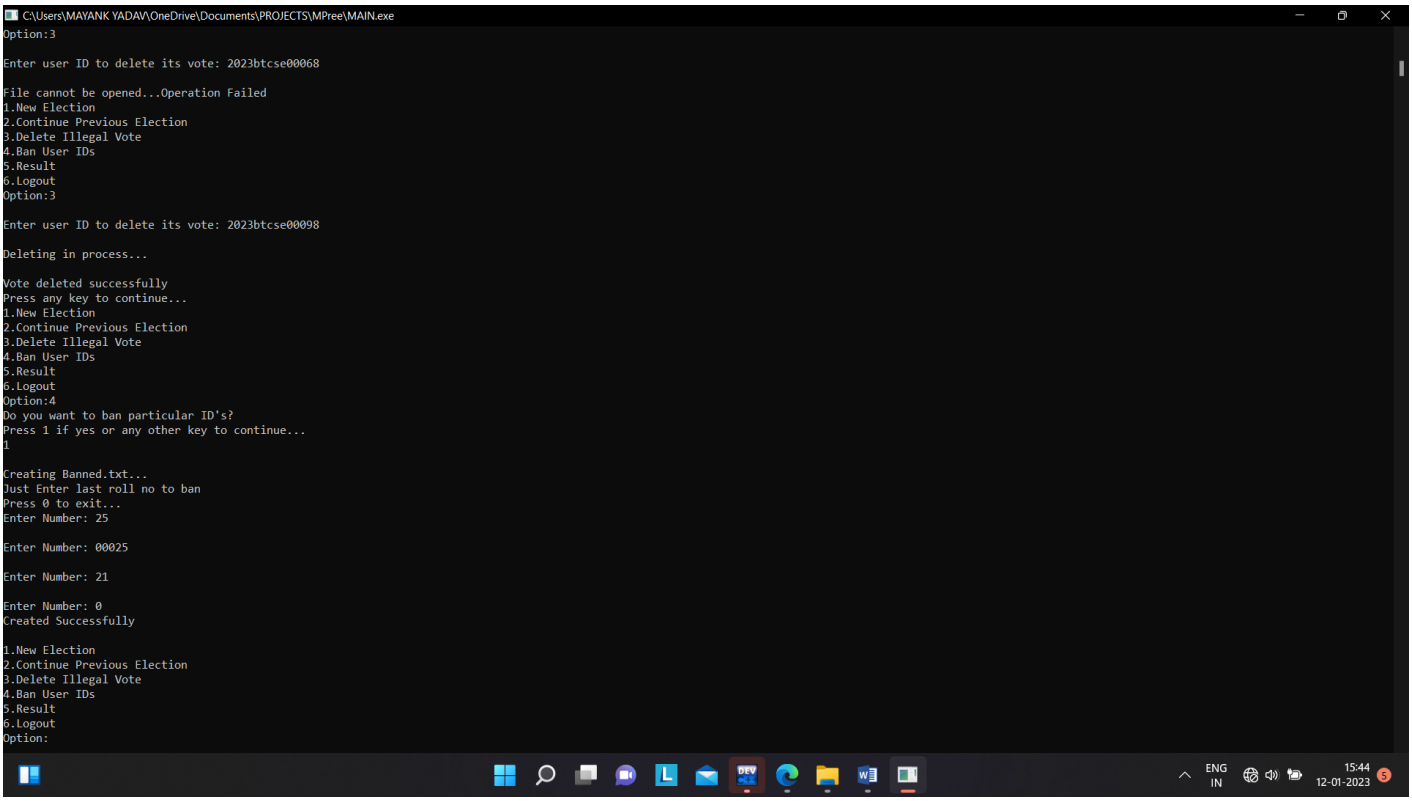
8.11 Delete a Vote



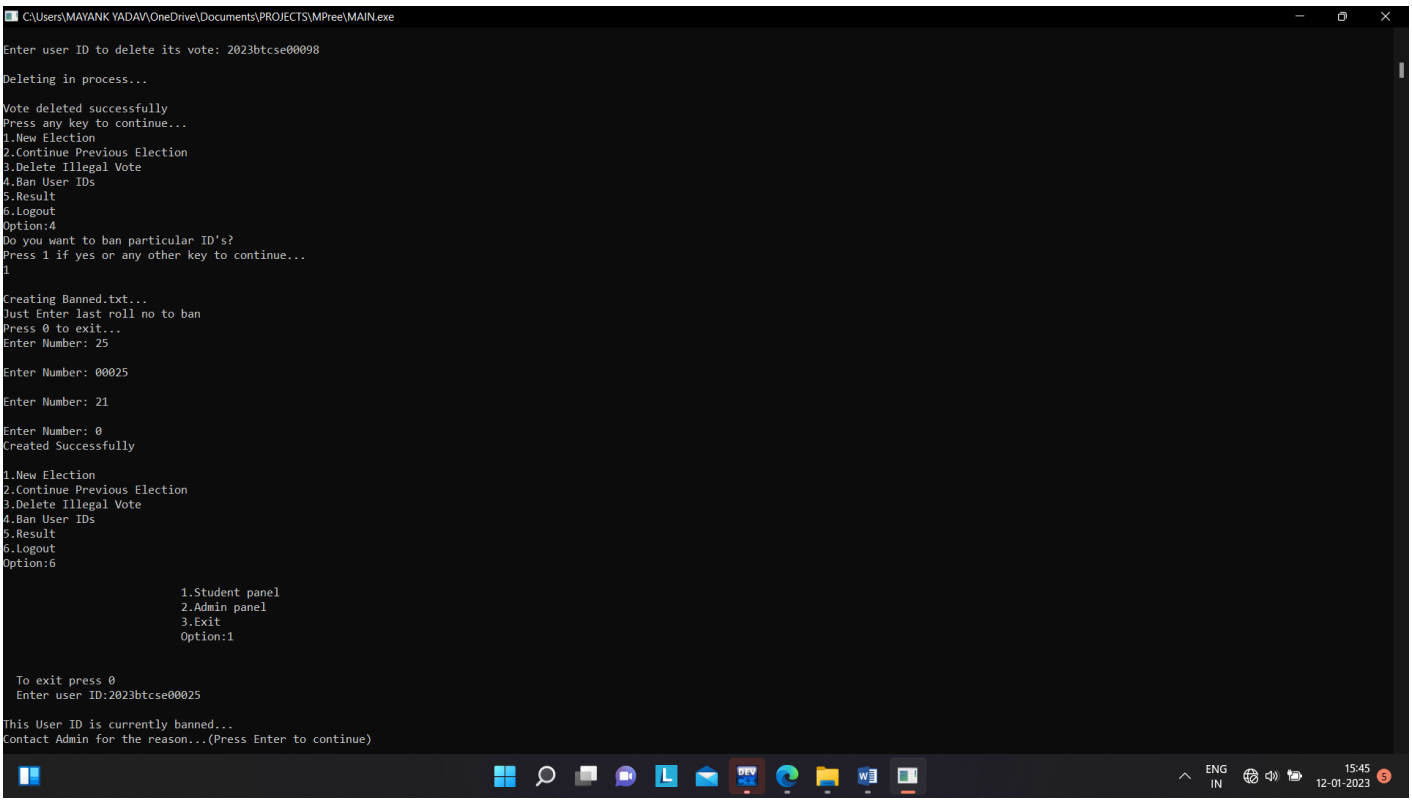
8.12 To Ban a vote



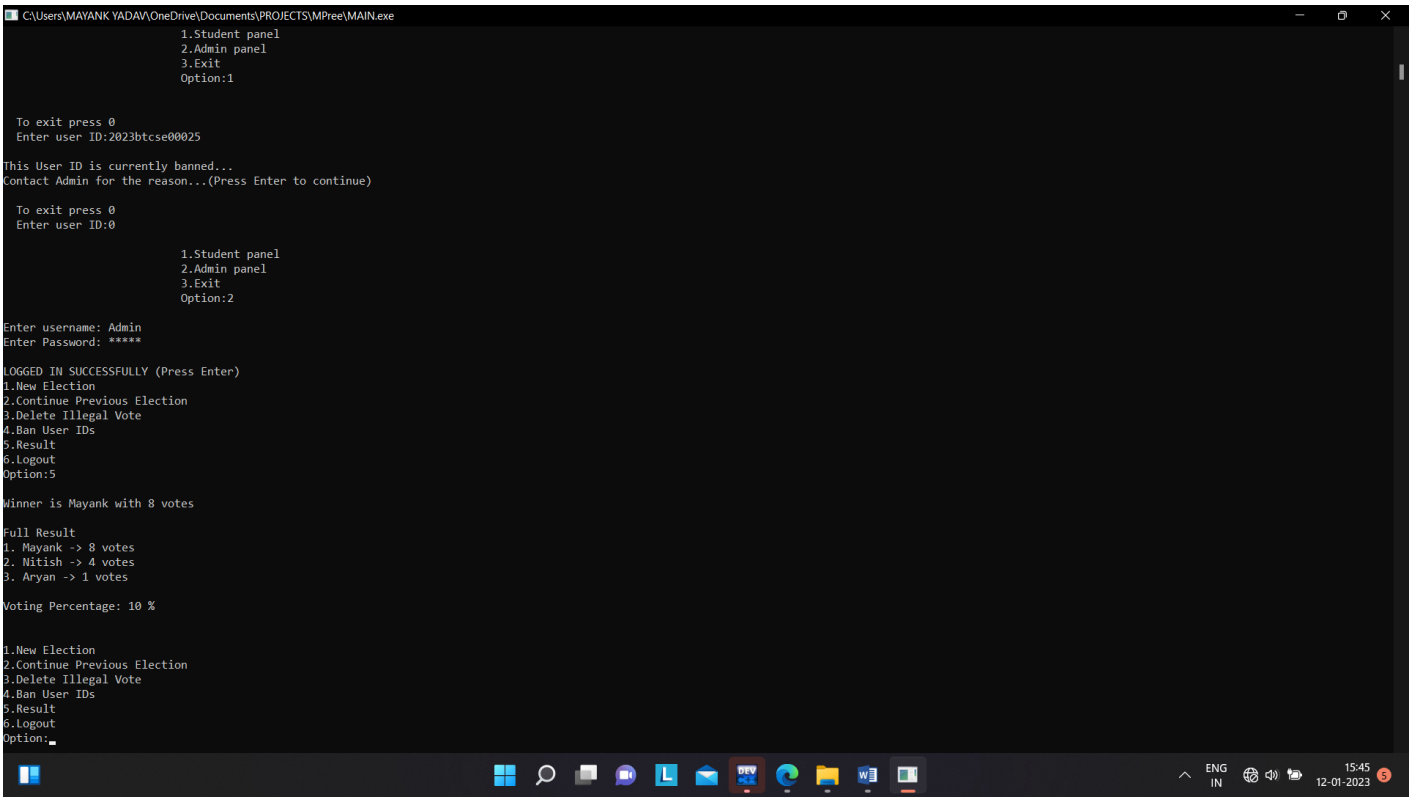
8.13 Ban user IDs



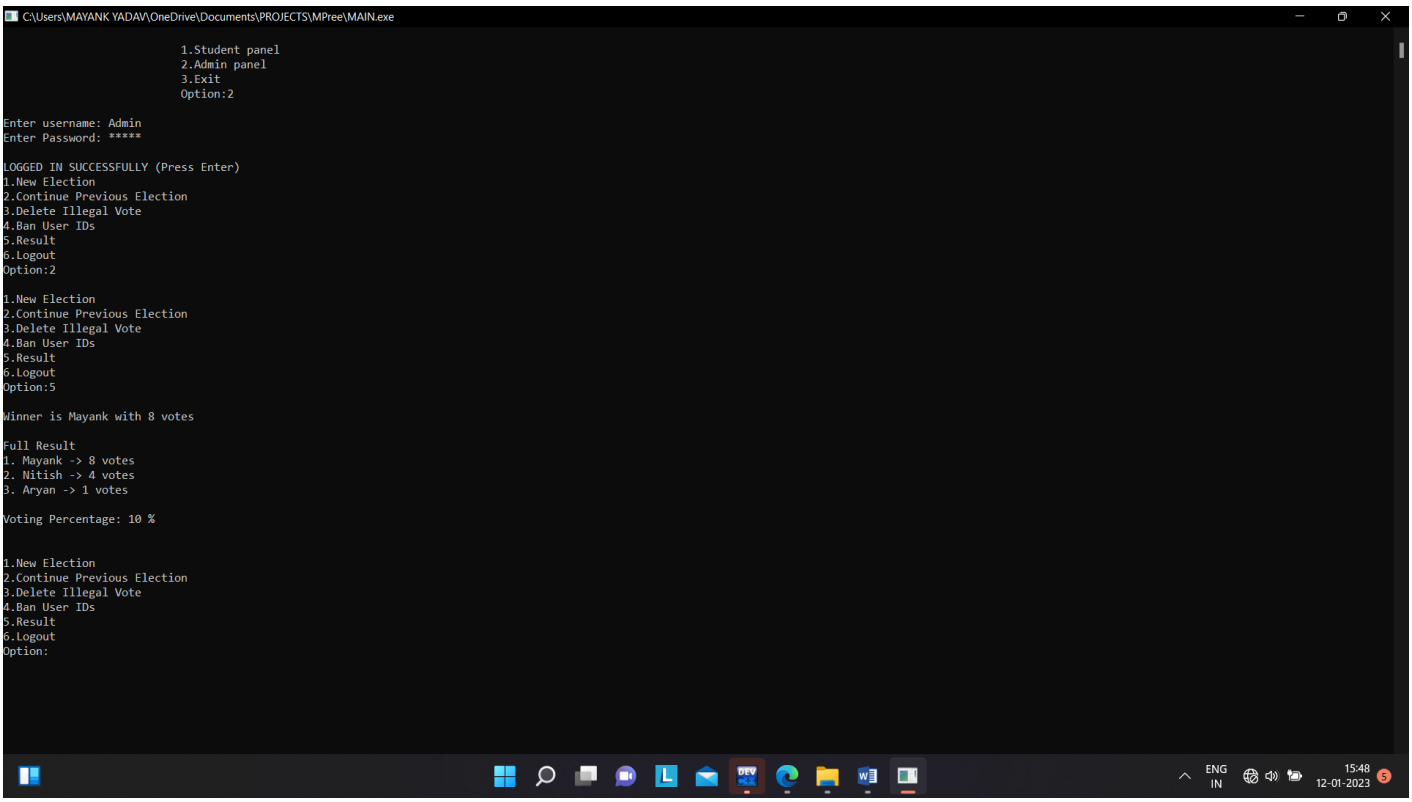
8.14 Restricting Banned User IDs



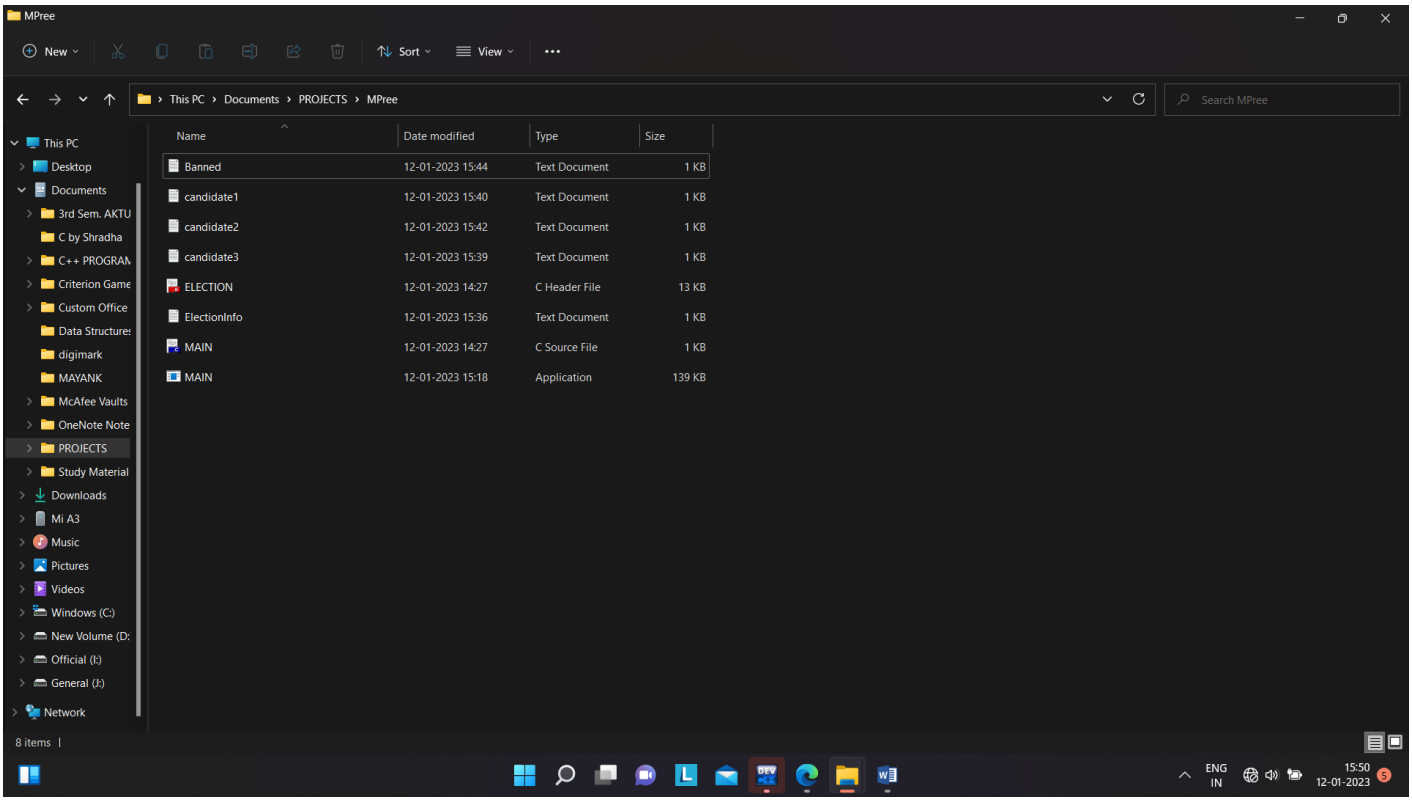
8.15 Result



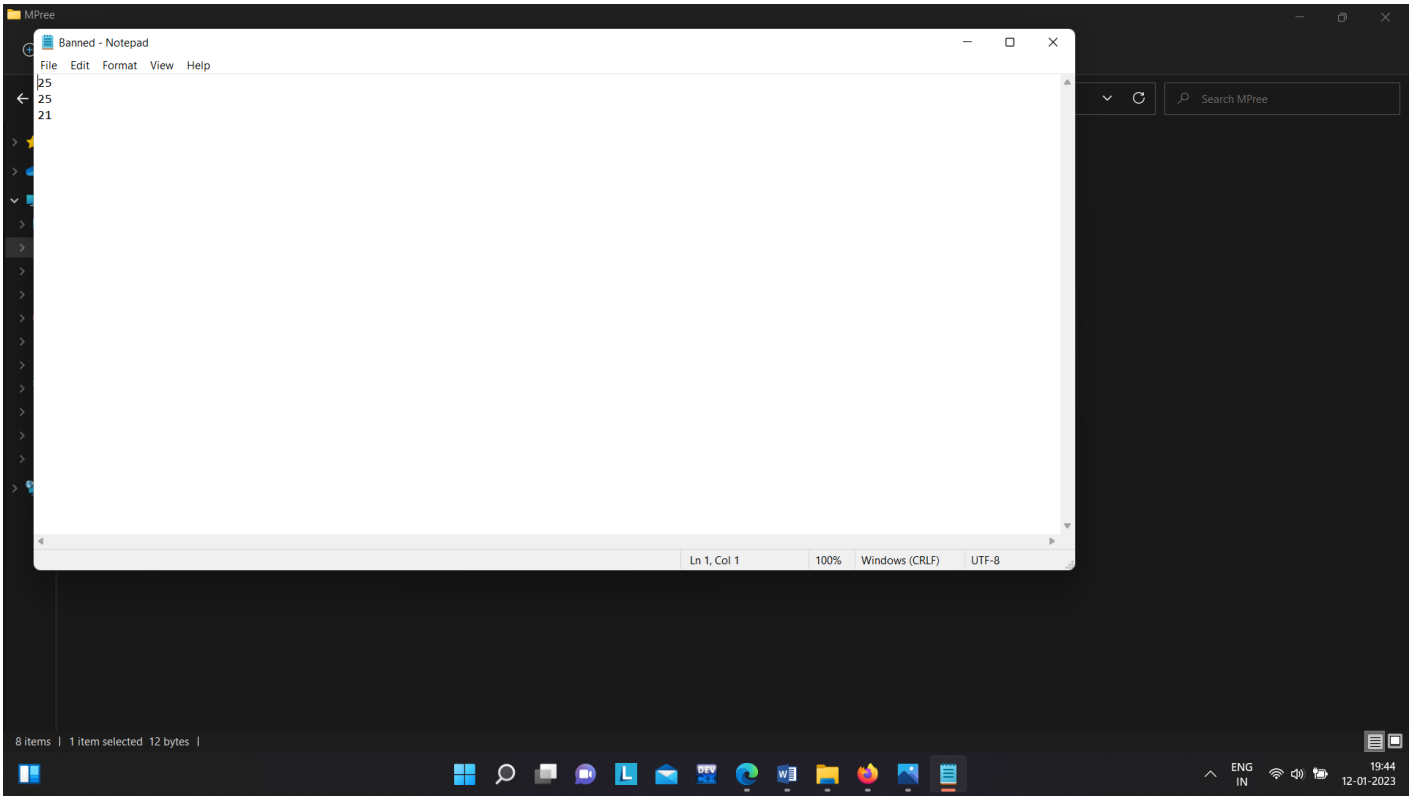
8.16 Previous Election Details



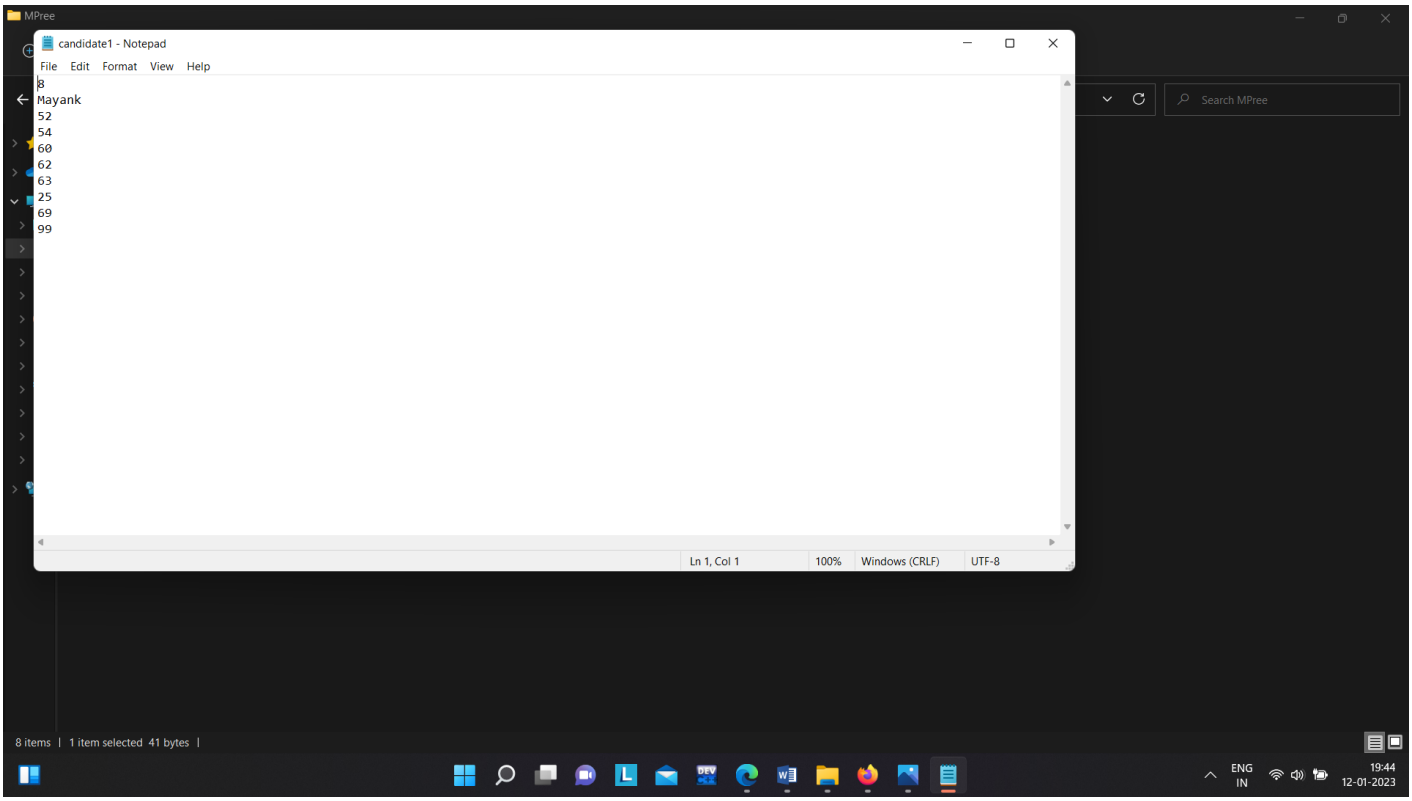
8.17 Details saved in This PC



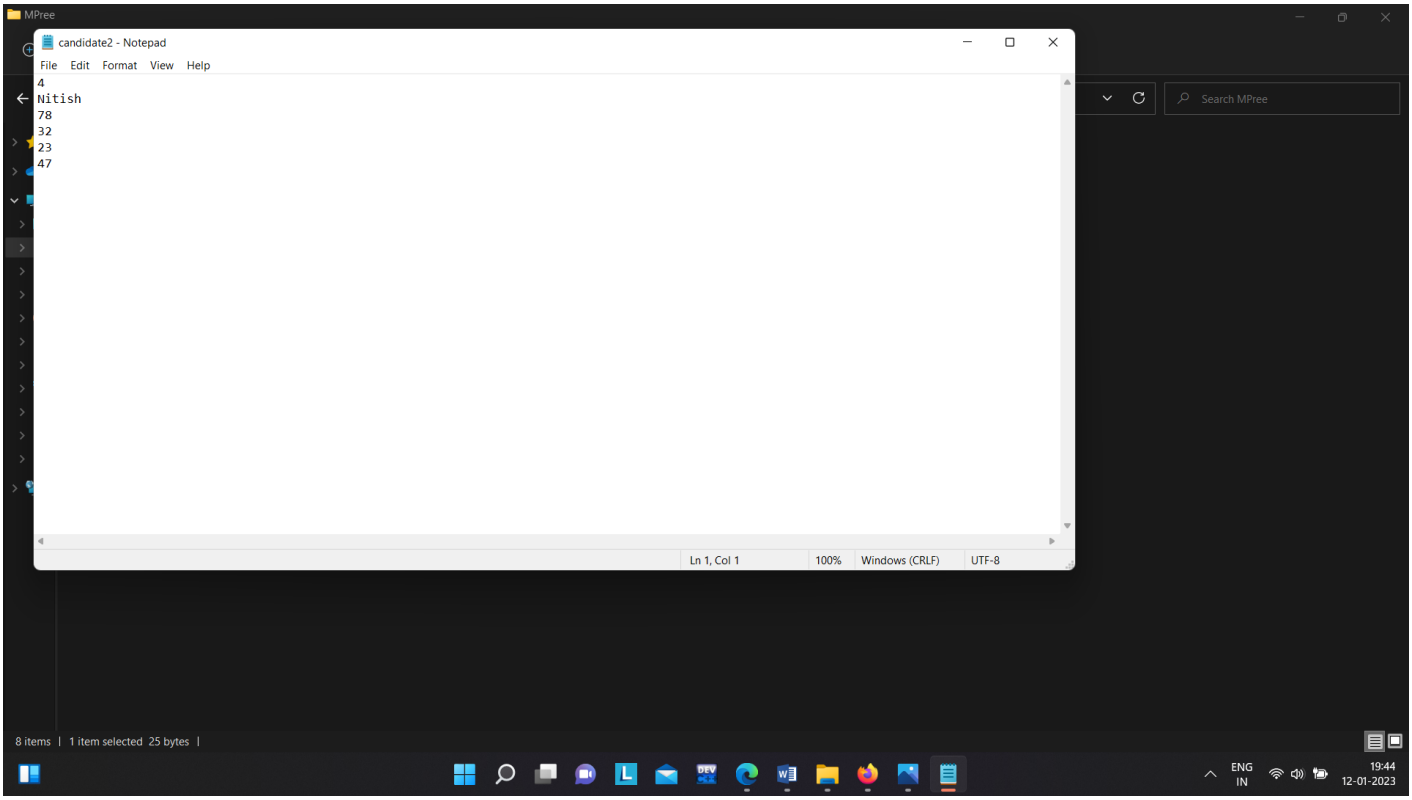
8.18 Banned voter IDs



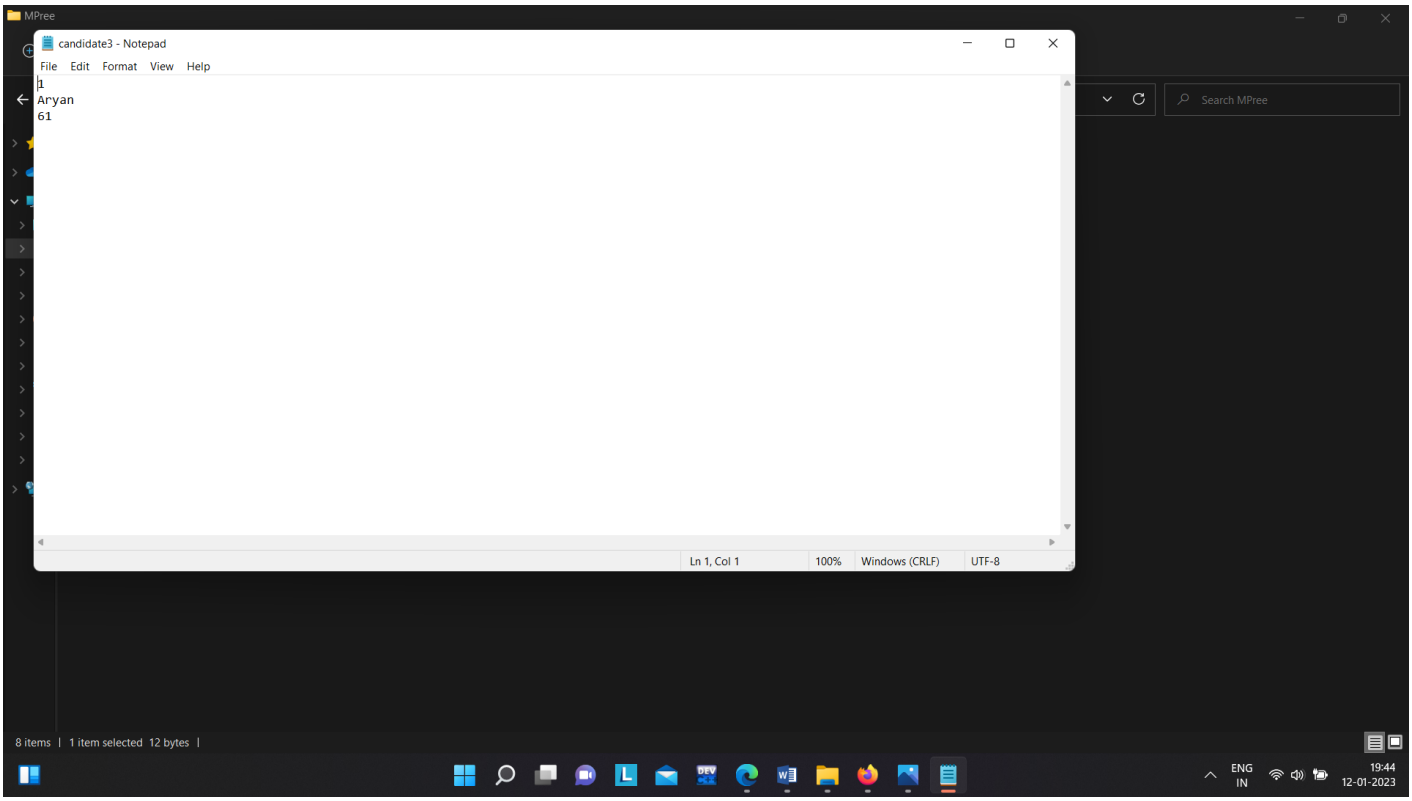
8.19 Candidate 1 votes



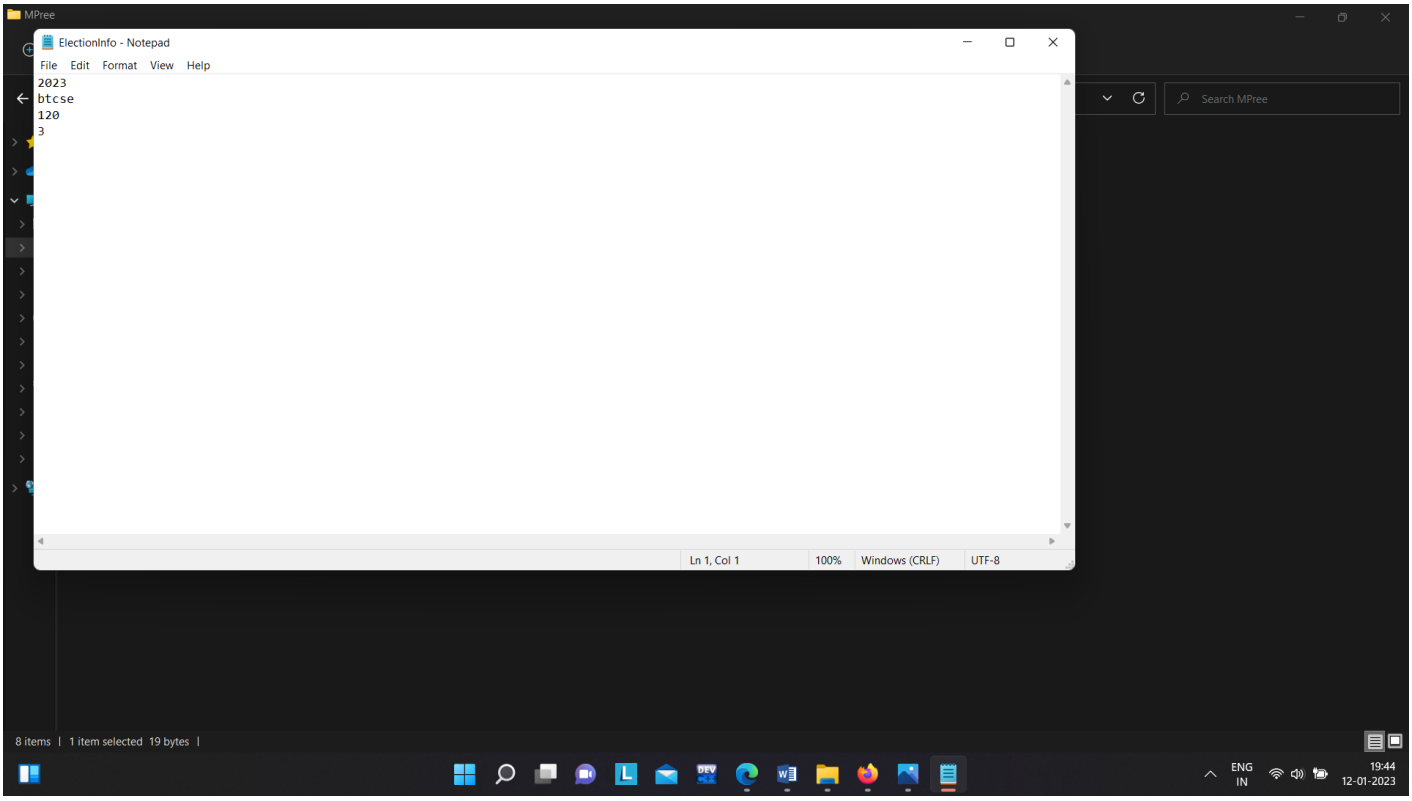
8.20 Candidate 2 votes



8.21 Candidate 3 votes



8.22 Election Information



9. Conclusion

As this website provides better way of election between voter and political parties: hence we suppose that this project as a greater scope and is important requirement is to provide a compact : stable system of voting with a facility through online.

This mini oting system will manage the Voter's information by which voter can login and use his voting rights. The system will incorporate all features of voting system. It provides the tools for maintaining voter's vote to every party and it count total no. of every party. There is a DATABASE which is maintained by the ELECTION COMMISION OF INDIA in which all the names of voter with complete information is stored.

In this user who is above 18years's register his/her information on the database and when he/she want to vote he/she has to login by his id and password and can vote to any party only single time. Voting detail store in database and the result is displayed by calculation. By online voting system percentage of voting is increases. It decreases the cost and time of voting process. It is very easy to use and it is very less time consuming.

It is very easy to debug.

The traditional method of manual voting system has few drawbacks. This method is obviously not efficient as it wastes the voter's energy and quite slow in term of completion. This smart system involves the voter's can cast their vote easily, and can be implemented to the entire India.

10. Future Enhancement

Data can be managed on cloud so that it will be secured and managed efficiently. We have developed the online system for only one particular booth , this should be extended to all the polling booths in India.

11. Bibliography

www.geekforgeeks.com

www.byjus.com/Introduction-To-C-Language

www.w3schools.com

https://en.wikipedia.org/wiki/Electronic_voting