Storage classes

class	Memory	Initial value	Scope	Life
Automatic	RAM	garbage	block	block
Register	Register	garbage	block	block
Static	RAM	0	block	program
Extern/global	RAM	0	program	program

```
Int main(){
        Static Int a=1;
        // print a
        A++;
        If A>100 return;
        welcome();
        main();
}

welcome(){printf(“welcome”,a);}
```

Bss, stack, heap

Searching, Sorting- complexity

	worst	best	Best case scenario-
Linear search	O(n)	O(1)	First element
Binary search	O(logn)	O(1)	Mid element
bubble	O(n^2)	O(n)	Sorted element
selection	O(n^2)	O(n)	Sorted elements
insertion	O(n^2)(inverse sorted)	O(n)	Sorted elements
quick	O(n^2) (sorted array)	O(nlogn) 1.6*O(nlog n)	Pivot is breaking array from the mid
merge	O(nlogn)	O(nlogn)	O(n)-->extra space
heap	O(nlogn)	O(nlogn)	Complex algo

Complexity- CPU time, RAM; time complexity, space complexity

Absolute time: X

Loop- number of elements-n

n(n+1)/2⇒n^2/2 +n/2⇒O(n^2)

O(1), O(logn), O(n), O(nlogn), O(n^m),O(2^n)

N/2^0, n/2^1, n/2^2,n/^3—----------n/2^k ===>2^k=n⇒k=logn

for(i=0;i<n-1;i++){
flag=0;

```
    for(j=0;j<n-1;j++){


}

}
```

STL sort()---> insertion, quick, heap

Pivot →
Partition strategy —>O(n)
O(n^2) O(nlogn)

1. Create a text editor in C++ with the following functionality:

a. Create storage for your data
b. Insert Data
c. Update Data
d. Append Data
e. Search Data
f. Delete data
g. Display Data on console
h. Undo operation
i. Redo operation
j. Save data in permanent storage

Overview

The program has CUI/CLI based commands. The programmer should choose data storage judiciously which is easy, efficient and effective for implementation. In the text editor, Insert data will be adding a unit of data depending on the data structure chosen. Update will replace the whole unit of data or a part of it depending on the requirement. Search will return the address or index of the searched element depending on the data structure. Delete will first find out the data

and then it will delete.Undo and Redo data will be for 5 previous operations only. Save data will store the data in the file so that it can be restored.

Once upon a time, in the world of programming, there was a young coder named Alex, eager to embark on a journey to create a powerful text editor in C++. Armed with ambition and a keen eye for innovation, Alex set out to conquer the challenge of building a feature-rich text editor from scratch.

Chapter 1: Inception

The idea sprouted in Alex's mind: a text editor with robust functionalities, a command-line interface (CUI), and a judiciously chosen data storage system. Excitedly, Alex outlined the essential functionalities: storage management, data manipulation, search, deletion, and the ability to undo and redo recent actions.

Chapter 2: Blueprint

With a clear vision, Alex delved into designing the architecture. After careful consideration, a dynamic array-based approach was chosen for data storage. The structure allowed flexibility while managing data efficiently. Alex coded the initial framework, creating functions to initialize, store, and manage data units.

Chapter 3: Implementing Functionality

The heart of the text editor began to take shape. Alex meticulously coded the functionalities, starting with the core operations:

- Insert Data: A function to add data units, dynamically resizing the storage if needed.
- Update Data: Capable of modifying entire data units or specific sections.
- Append Data: Adding data at the end of the storage structure.
- Search Data: Utilizing algorithms to find and return the index or address of the searched element.
- Delete Data: First locating the data and then eliminating it efficiently from the structure.

Chapter 4: User Interaction

The command-line interface was crafted, offering users a seamless experience. Alex implemented a menu-based system, allowing users to input commands for various operations. The console displayed data and prompted users for further actions.

Chapter 5: Undo, Redo, and Save

Alex faced an intriguing challenge—enabling the editor to undo and redo operations. With meticulous planning, Alex created a mechanism to store the last five actions, allowing users to backtrack or redo recent changes. Additionally, a function was built to save the data onto a file, ensuring persistence across sessions.

Chapter 6: Testing and Debugging

With the editor's framework complete, Alex rigorously tested every functionality, identifying and fixing bugs. User scenarios were simulated, ensuring smooth operation and error handling.

Chapter 7: The Text Editor Awakens

As the final lines of code were written, the text editor sprung to life. Alex executed the program, witnessing the fruits of labor—data insertion, updates, searches, deletions—all flawlessly executed through the carefully crafted CUI. The editor allowed users to undo, redo, and save their work, ensuring a seamless editing experience.

With a sense of accomplishment, Alex marveled at the editor—a testament to determination, coding prowess, and a passion for innovation.

And thus, a remarkable text editor, born from imagination and dedication, stood ready to empower users in the vast landscape of data and text manipulation.

Line1 :The program has CUI/CLI based commands.
Line2: The programmer should choose data storage judiciously which is easy, efficient and effective for implementation.
Line3:

Line 4: In the text editor, Insert data will be adding a unit of data depending on the data structure chosen.
Update will replace the whole unit of data or a part of it depending on the requirement.
Search will return the address or index of the searched element depending on the data structure.
Delete will first find out the data and then it will delete.Undo and Redo data will be for 5 previous operations only.
Save data will store the data in the file so that it can be restored.

Line 2, replace, insertion, deletion