Data Structures Assignment for CAT1 by ***Sai Rahul***

3122215001090 CSEB

Find a suitable data structure to be used to solve the following problem. [CO6, K4]

[2.2.4, 3.2.1]

Assume that ADT for the chosen data structure is available. Use the ADT to write a complete program in C to implement **a simple online text editor** with the following operations in a modular way. The editor initially contains an empty string S and user types a string of alphabets and a command 1, 2, 3 and 4 for the operations: [CO2, K3]

[2.1.2, 2.3.1, 12.3.1, 13.3.1, 13.3.2, 1.4.1, 4.1.2]

1. *Append - Append string to the end of S*
2. *Delete - Delete the last K characters of S*
3. *Print - Print the last K characters of S*
4. *Undo - Undo the last operation (1 or 2), reverting to the state it was in prior to that operation. Assume that 2 consecutive undo's are not allowed.*

Prepare a report containing the following: [10.1.1, 10.1.2]

i. Exploration of Design Alternatives [3]
ii. Rationale between choices [3]
iii. Identify modules with their prototypes in the program [2]
iv. Choose and write appropriate ADT procedures to implement modules [12]
v. Program with Output snapshot [5]

Example:

S="abcde"

No.	Sentence (S)	Command	Explanation
1	abcde	1 fg	Append fg to the end of S
2	abcdefg	3 3	Print the last 3 characters efg
3	abcdefg	2 5	Delete last 5 characters
4	ab	4	Undo the last operations
5	abcdefg	1 hi	Append hi to the end of S
6	abcdefghi	3 3	Print the last 3 characters ghi

Assignment in data structures:

i. Exploration of Design Alternatives

Data alternatives include doing the same task with the help of

- linked lists
- arrays
- queues
- lists
- instead of using stacks(which I have used)

ii)Rationale between choices

- I have used stacks as it has the push pop operations which are very useful in the code.other data structures don't have these kind of operations which makes the programming easier for the above scenario when I use stacks
- Stacks are useful and here in this scenario the order of insertion is important this is because they ensure that a system does not move onto a new action before completing those before.
- Stacks have advantages over other data structures like say for example arrays

iii) ***Identify modules with their prototypes in the program***

void append(char* ch)

input:the character ch(the word)

output:the appended word

the type is:void

the argument is:char*ch

void erase(int k)

the type is void

the argument is int k

input:the no of lettrs to be deleted (current)

output:the new string after the requires no of letters have been removed

void print()

the type is void

there is no argument

the input is nothing(it takes the character as it is as the input)

output is: the string as it is

printf("%s",peek())

void undo()

input:there is no input as such

output:the previous operation is undone

the typre is void

the arguments are none

<u>The stack has some functions:</u>

int is_empty()

checks If the stack is empty:

type is int

arguments are none

input:we don’t give any input

output:tells True if empty False if non empty

int is_full()

checks if the stack is full

type:int

arguments:none

input:no input given uses the string as input

output:returns if the stack is full or not

void push(char* ch)

pushes the character into the stack

type void

arguments:char*ch

input:the ch character is the input which is pushed

output:the new stack with the elements pushed into it

char* peek()

peek() helps to retrieve the topmost element in the stack

type:char

arguments:none

inputs:the stack(not given directly)

outputs:

the topmost element

char* pop()

pops out characters from the stack

type:char

arguments none

inputs:none

output:the popped element

iv) Choose and write appropriate ADT procedures to implement modules

void append(char* ch)

input:the character ch(the word) it is taken as arguement

output:the appended word

the type is:void

the argument is:char*ch

void erase(int k)

the type is void

the argument is int k

input:the no of lettrs to be deleted (current)

output:the new string after the requires no of letters have been removed

void print()

the type is void

there is no argument

the input is nothing(it takes the character as it is as the input)

output is: the string as it is

printf("%s",peek())

void undo()

input: there is no input as such

output: the previous operation is undone

the type is void

the arguments are none

The stack has some functions:

int is_empty()

checks If the stack is empty

type is int

arguments are none

input: we don't give any input

output: tells True if empty False if non empty

int is_full()

checks if the stack is full

type:int

arguments:none

input:no input given uses the string as input

output:returns if the stack is full or not

void push(char* ch)

pushes the character into the stack

type void

arguments:char*ch

input:the ch character is the input which is pushed

output:the new stack with the elements pushed into it

void get(int k,char*ch)

input:does not explicitly take input uses current as the input

output: gets the characters

type:void

arguments:int k,char*ch

char* peek()

is to retrive the topmost element in the stack

type:char

arguments:none

inputs:the stack(not given directly)

outputs:

the topmost element

char* pop()

pops out characters from the stack

type:char

arguments none

inputs:none

output:the popped element

The ADT Procedures used in the program are append delete print ands undo

The method ***void append(char* ch)*** adds a string to the pre existing string

void erase() deletes the no of characters we have mentioned

void print() method displays the last K elements,K is taken as input from the user

void undo() method undoes the last operation

The Stack in itself is a method which contains many operations eg

int is_empty()

checks If the stack is empty:

Algorithm:

now defining int is_empty() function

```
return (top < 0)
```

this returns if the stack is empty or not

int is_full()

checks if the stack is full

Algorithm:

now defining int is_full() function

```
return (top > MAX)
```

this returns true if full

void push(char* ch)

pushes the character into the stack

Algorithm:

now defining void push(char* ch)

```
if (is_full() does not hold true) =>
    stack[++top] = ch
```

char* peek()

is to retrive the topmost element in the stack

Algorithm:

char* peek() =>we are defining the fn

  char* top_data= is set to '\0'

  if (!is_empty())

    top_data = stack[top] means the top of the stack

  return top_data

char* pop()

pops out characters from the stack

Algorithm:

char* pop()

  char* top_data= peek()

  if (top_data) =>

    stack[top--] is set to '\0'

  return top_data

now writing algorithm for getlen

input the character/string(indirectly given)

o/p:the length

int get_len(char* ch)

  int len = 0

```
while(*ch) =>
    len++; ch++

return len;
```

Now writing algorithm for

void append(char ch)*

```
int i,len = get_len(ch)
char* current = peek()
if (not equals current)
    current = (char*) malloc(sizeof(char) * (len + 1))=>we are allocatinf some memory
    for (i = 0; i < len; i++)
    current[i] = ch[i]

    current[len] = '\0'
    push(current)
else
                int j is set to 0
    int current_len = get_len(current)
    char* current_new = (char*)malloc(sizeof(char) * (current_len + len + 1))

    if (current_new)
        for ( i = 0; i < current_len; i++)
            current_new[i] = current[i]

                        for ( i = current_len; i < current_len + len; i++)
                                current_new[i] = ch[j++]
```

```
        current_new[current_len + len] = '\0'
        push(current_new);
```

Now writing algorithm for

void erase(int k) {

```
   char* current = peek()=>peek() means top of the stack
   int i
   if (current)
      int current_len = get_len(current)
      if (current_len >= k)
         char* current_new = (char*)malloc(sizeof(char) * (current_len - k + 1))=>we are allocating
memory
         if (current_new)
            for (i = 0; i < current_len - k; i++)
               current_new[i] = current[i]

            current_new[current_len - k] = '\0'
            push(current_new)
```

now writing algorithm for:

```
void print()
 printf("\nTHE SENTENCE  S : ")
           int i
           printf("%s",peek())
           printf("\n")
```

<u>Now defining the next function</u>

```
void get(int k, char* ch)
   char* current = peek()

   if (current)
      int current_len = get_len(current)
      if (current_len >= k)
         *ch = current[k - 1]
```

Now writing algorithm for:

```
void get(int k, char* ch)
   char* current = peek()

   if (current) =>
      int current_len = get_len(current)
      if (current_len >= k)
         *ch = current[k - 1]
```

Now defining the undo function

```
void undo()
   pop()
```

now writing the program to display the utility of all the above methods I have used

//I have chosen stack datatype due to stacks advantages

//code

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#define MAX 1000

char* stack[MAX];
int top = -1;

int is_empty() {
   return (top < 0);
}

int is_full() {
   return (top > MAX);
}

void push(char* ch) {
   if (!is_full()) {
      stack[++top] = ch;
   }
}

char* peek() {
   char* top_data= '\0';
   if (!is_empty()) {
      top_data = stack[top];
   }
   return top_data;
}
```

```
char* pop() {
   char* top_data= peek();
   if (top_data) {
      stack[top--] = '\0';
   }
   return top_data;
}

int get_len(char* ch) {
   int len = 0;
   while(*ch) {
      len++; ch++;
   }
   return len;
}
void append(char* ch) {
   int i,len = get_len(ch);
   char* current = peek();
   if (!current) {
      current = (char*) malloc(sizeof(char) * (len + 1));
      for (i = 0; i < len; i++) {
      current[i] = ch[i];
      }
      current[len] = '\0';
      push(current);
   } else {
                        int j = 0;
      int current_len = get_len(current);
      char* current_new = (char*)malloc(sizeof(char) * (current_len + len + 1));
```

```
        if (current_new) {
            for ( i = 0; i < current_len; i++) {
                current_new[i] = current[i];
            }
                                for ( i = current_len; i < current_len + len; i++) {
                                        current_new[i] = ch[j++];
                                }
            current_new[current_len + len] = '\0';
            push(current_new);
        }
    }
}
```

void erase(int k) {

```
    char* current = peek();
    int i;
    if (current) {
        int current_len = get_len(current);
        if (current_len >= k) {
            char* current_new = (char*)malloc(sizeof(char) * (current_len - k + 1));
            if (current_new) {
                for (i = 0; i < current_len - k; i++) {
                    current_new[i] = current[i];
                }
                current_new[current_len - k] = '\0';
                push(current_new);
            }
        }
    }
}
```

void print()

```
{ printf("\nTHE SENTENCE  S : ");
            int i;
            printf("%s",peek());
            printf("\n");
}
void get(int k, char* ch) {
   char* current = peek();

   if (current) {
      int current_len = get_len(current);
      if (current_len >= k) {
         *ch = current[k - 1];
      }
   }
}

void undo() {
   pop();
}

int main() {

   int op,i,choice=1;
   int k,l=0,p=0;
   char data[MAX];
   char *s="abcde";
   append(s);
   print();
   do
   {
                        printf("\nEnter choice :(in format [option arguments])\n");
```

```
        printf("\n1. Append String to end of S");

                printf("\n2. Delete last k charcters to S");
                printf("\n3. Print Last K characters of S");
        printf("\n4. Undo");
        printf("\n5. Exit");
 scanf("%d", &op);
switch(op)
{
            case 1: scanf("%s", data);
                append(data);
                print();
                break;
              case 2:
           scanf("%d", &k);
           erase(k);
           print();
           break;
                    case 3:
                        scanf("%d", &k);
                        l=strlen(stack[top])-1;
                          printf("\nlast %d characters : ",k);

                        for(p=l-k+1;p<=l;p++)
                        printf("%c",stack[top][p]);
                        printf("\n");
           break;
     case 4:
        undo();
        print();
        break;
```

```
        case 5: exit(0); break;
        default: printf("\nInvalid Choice"); break;
      }

      }while(op);


   return 0;
}
```

Algorithm/Pseudocode:

Input:the operation we want to perform(this is inputted in the form of the number)

Output:The required operations are performed on the stack

<u>Algorithm for entire code</u>

Include the nessessary packages:

include <stdio.h> package

include <string.h> Package

include <math.h> package

include <stdlib.h> package

define MAX 1000

char* stack[MAX] we are initializing

int top is initialized to -1

Now we are defining the functions

now defining int is_empty() function

    return (top < 0)

now defining int is_full() function

    return (top > MAX)

now defining void push(char* ch)

    if (is_full() does not hold true) =>

        stack[++top] = ch

char* peek()

    char* top_data= is set to '\0'

    if (!is_empty())

        top_data = stack[top] means the top of the stack

    return top_data

char* pop()

    char* top_data= peek()

    if (top_data) =>

        stack[top--] is set to '\0'

    return top_data

```
int get_len(char* ch)
   int len = 0
   while(*ch) =>
      len++; ch++

   return len;

void append(char* ch)
   int i,len = get_len(ch)
   char* current = peek()
   if (not equals current)
      current = (char*) malloc(sizeof(char) * (len + 1))=>we are allocatinf some memory
      for (i = 0; i < len; i++)
      current[i] = ch[i]

      current[len] = '\0'
      push(current)
   else
                     int j is set to 0
      int current_len = get_len(current)
      char* current_new = (char*)malloc(sizeof(char) * (current_len + len + 1))

      if (current_new)
         for ( i = 0; i < current_len; i++)
            current_new[i] = current[i]

                              for ( i = current_len; i < current_len + len; i++)
                                       current_new[i] = ch[j++]

         current_new[current_len + len] = '\0'
```

```
        push(current_new);
```

Now we are defining the below function

```
void erase(int k) {
   char* current = peek()=>peek() means top of the stack
   int i
   if (current)
      int current_len = get_len(current)
      if (current_len >= k)
         char* current_new = (char*)malloc(sizeof(char) * (current_len - k + 1))=>we are allocating
memory
         if (current_new)
            for (i = 0; i < current_len - k; i++)
               current_new[i] = current[i]

            current_new[current_len - k] = '\0'
            push(current_new)
```

now defining the next function

```
void print()
 printf("\nTHE SENTENCE  S : ")
           int i
           printf("%s",peek())
           printf("\n")
```

Now defining the next function

```
void get(int k, char* ch)
   char* current = peek()

   if (current)
      int current_len = get_len(current)
      if (current_len >= k)
         *ch = current[k - 1]
```

Now defining the undo function

```
void undo()
   pop()
```

now going to the main program

```
int main()

   int op,i,choice=1
   int k,l=0,p=0
   char data[MAX]
   char *s="abcde"
   append(s)
   print()
   do(we are using do while loop)

                  print("\nEnter choice :(in format [option arguments])\n")
         print("\n1. Append String to end of S")

                  print("\n2. Delete last k charcters to S")
                  print("\n3. Print Last K characters of S")
         print("\n4. Undo")
         print("\n5. Exit")
```

```
 scan(op)
switch(op)=>we are using switch case

          case 1: scanf("%s", data)
                append(data)
                print()
                break
          case 2:
        scan(k)
        erase(k)
        print()
        break
                 case 3:
                     scan(k)
                     l=strlen(stack[top])-1
                       printf("\nlast %d characters : ",k)

                     for(p=l-k+1;p<=l;p++)
                     printf("%c",stack[top][p])
                     printf("\n")
        break
   case 4:
   we are calling the functions
      undo()
      print()
      break;
    case 5: exit(0); break
    default: printf("\nInvalid Choice"); break


 while(op)=>now the do while loop ends
```

```
return 0
```

5)

Sreenshots of the output and the program

```
//I have chosen stack datatype
//code
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#define MAX 1000
```

```
char* stack[MAX];
int top = -1;

int is_empty() {
   return (top < 0);
}

int is_full() {
   return (top > MAX);
}

void push(char* ch) {
   if (!is_full()) {
      stack[++top] = ch;
   }
}

char* peek() {
   char* top_data= '\0';
   if (!is_empty()) {
      top_data = stack[top];
   }
   return top_data;
}

char* pop() {
   char* top_data= peek();
   if (top_data) {
      stack[top--] = '\0';
   }
   return top_data;
```

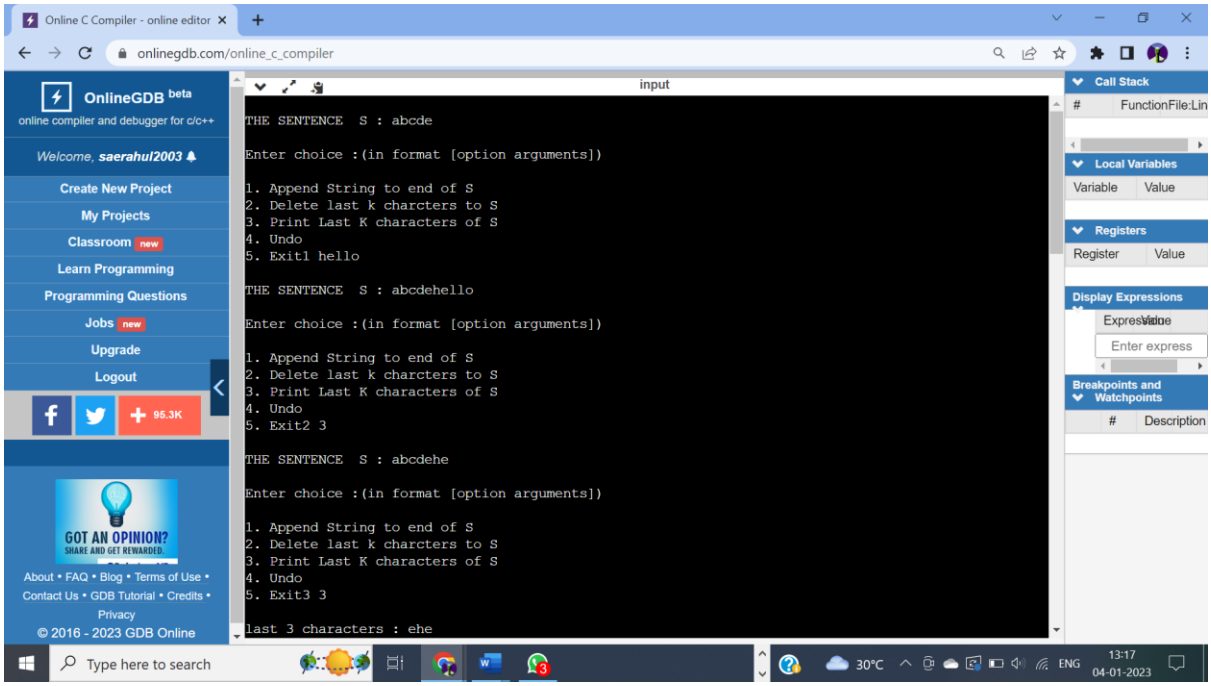
```
}

int get_len(char* ch) {
   int len = 0;
   while(*ch) {
      len++; ch++;
   }
   return len;
}
void append(char* ch) {
   int i,len = get_len(ch);
   char* current = peek();
   if (!current) {
      current = (char*) malloc(sizeof(char) * (len + 1));
      for (i = 0; i < len; i++) {
      current[i] = ch[i];
      }
      current[len] = '\0';
      push(current);
   } else {
                        int j = 0;
      int current_len = get_len(current);
      char* current_new = (char*)malloc(sizeof(char) * (current_len + len + 1));

      if (current_new) {
         for ( i = 0; i < current_len; i++) {
            current_new[i] = current[i];
         }
                                 for ( i = current_len; i < current_len + len; i++) {
                                          current_new[i] = ch[j++];
                                 }
```

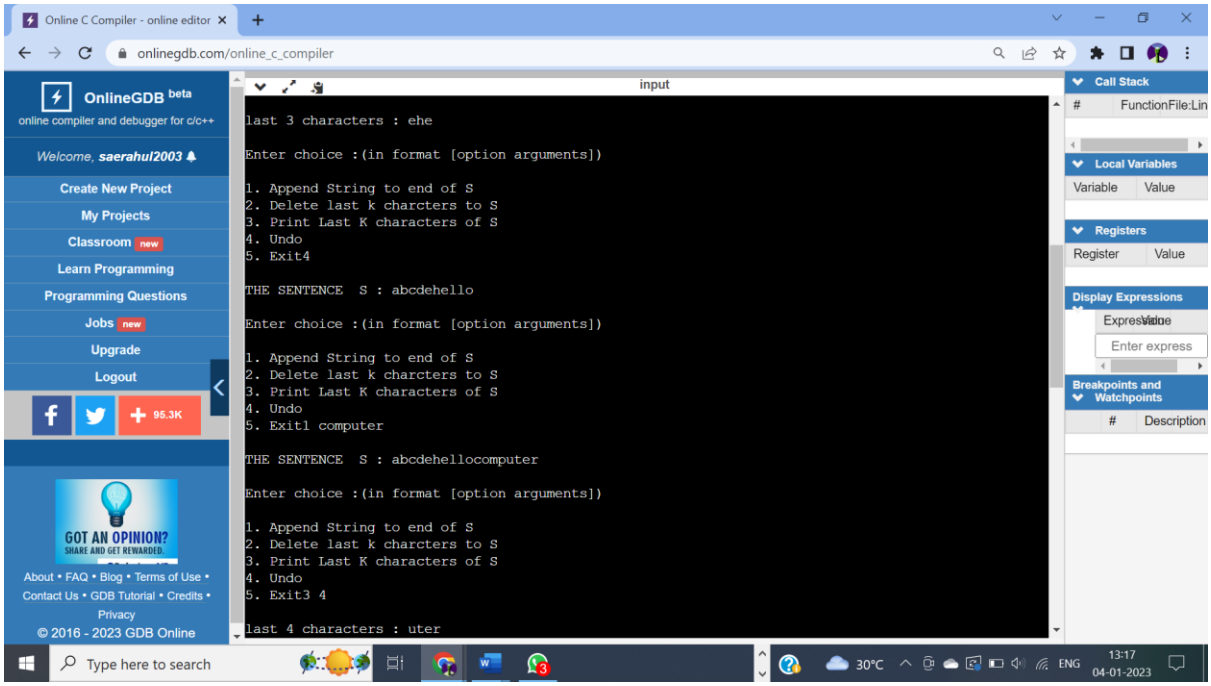
```
            current_new[current_len + len] = '\0';
            push(current_new);
        }
    }
}

void erase(int k) {
    char* current = peek();
    int i;
    if (current) {
        int current_len = get_len(current);
        if (current_len >= k) {
            char* current_new = (char*)malloc(sizeof(char) * (current_len - k + 1));
            if (current_new) {
                for (i = 0; i < current_len - k; i++) {
                    current_new[i] = current[i];
                }
                current_new[current_len - k] = '\0';
                push(current_new);
            }
        }
    }
}
void print()
{ printf("\nTHE SENTENCE  S : ");
              int i;
              printf("%s",peek());
              printf("\n");
}
void get(int k, char* ch) {
    char* current = peek();
```

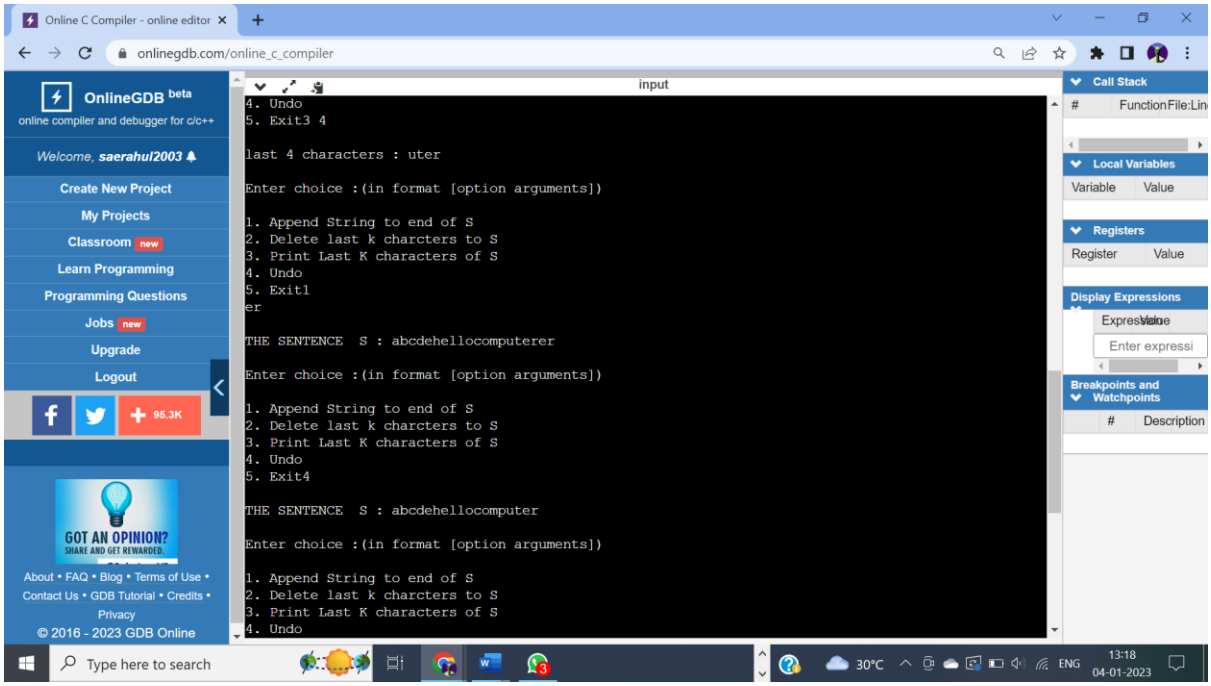
```
    if (current) {
        int current_len = get_len(current);
        if (current_len >= k) {
            *ch = current[k - 1];
        }
    }
}

void undo() {
    pop();
}

int main() {

    int op,i,choice=1;
    int k,l=0,p=0;
    char data[MAX];
    char *s="abcde";
    append(s);
    print();
    do
    {
                printf("\nEnter choice :(in format [option arguments])\n");
            printf("\n1. Append String to end of S");

                printf("\n2. Delete last k charcters to S");
                printf("\n3. Print Last K characters of S");
            printf("\n4. Undo");
            printf("\n5. Exit");
        scanf("%d", &op);
```

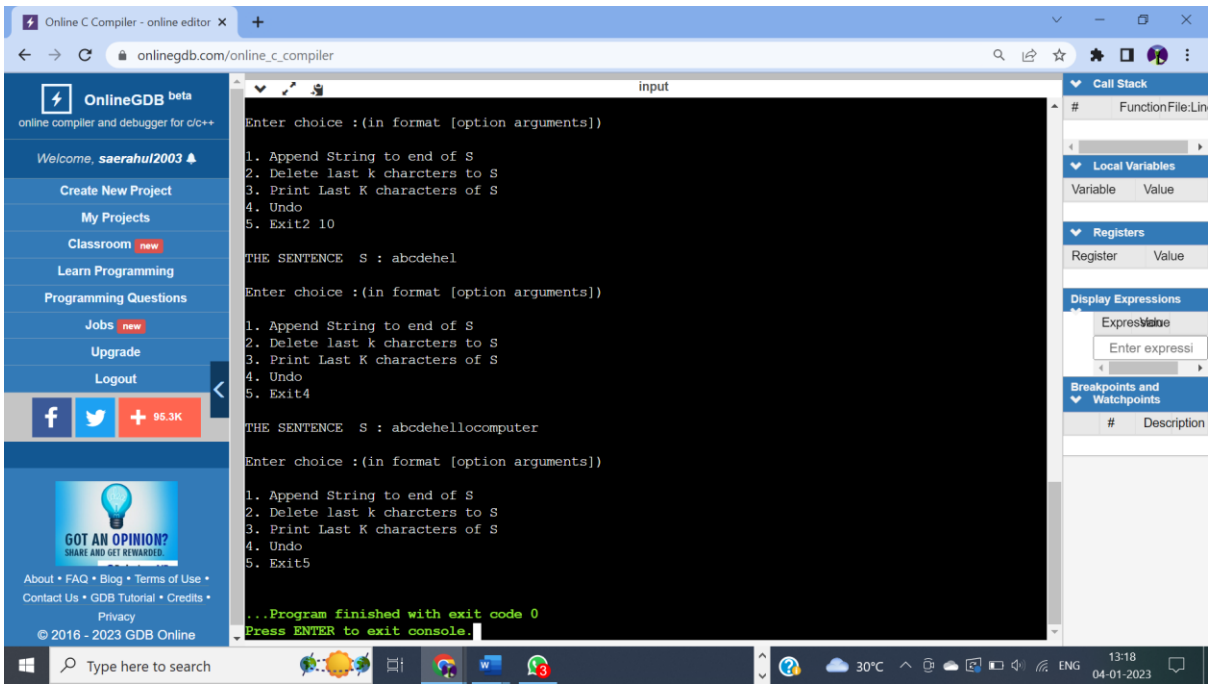
```
switch(op)
{
            case 1: scanf("%s", data);
                    append(data);
                    print();
                    break;
             case 2:
           scanf("%d", &k);
           erase(k);
           print();
           break;
                     case 3:
                          scanf("%d", &k);
                          l=strlen(stack[top])-1;
                            printf("\nlast %d characters : ",k);

                          for(p=l-k+1;p<=l;p++)
                          printf("%c",stack[top][p]);
                          printf("\n");
           break;
     case 4:
         undo();
         print();
         break;
      case 5: exit(0); break;
      default: printf("\nInvalid Choice"); break;
 }

 }while(op);
```

```
    return 0;
}
```


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
OnlineGDB beta
online compiler and debugger for c/c++
Welcome, saerahul2003
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Upgrade
Logout
95.3K
GOT AN OPINION?
SHARE AND GET REWARDED
About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online
input
THE SENTENCE S : abcde
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit1 hello
THE SENTENCE S : abcdehello
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit2 3
THE SENTENCE S : abcdehe
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit3 3
last 3 characters : ehe
Call Stack
Local Variables
Registers
Display Expressions
Breakpoints and Watchpoints
Type here to search
30°C
ENG
13:17
04-01-2023


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
OnlineGDB beta
online compiler and debugger for c/c++
Welcome, saerahul2003
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Upgrade
Logout
95.3K
GOT AN OPINION?
SHARE AND GET REWARDED
About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online
input
last 3 characters : ehe
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit4
THE SENTENCE S : abcdehello
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit1 computer
THE SENTENCE S : abcdehellocomputer
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit3 4
last 4 characters : uter
Call Stack
Local Variables
Registers
Display Expressions
Breakpoints and Watchpoints
Type here to search
30°C
ENG
13:17
04-01-2023


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
OnlineGDB beta
online compiler and debugger for c/c++
Welcome, saerahul2003
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Upgrade
Logout
95.3K
GOT AN OPINION?
SHARE AND GET REWARDED.
About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online
input
4. Undo
5. Exit3 4
last 4 characters : uter
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit1
er
THE SENTENCE S : abcdehellocomputererer
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit4
THE SENTENCE S : abcdehellocomputer
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
Call Stack
Local Variables
Variable Value
Registers
Register Value
Display Expressions
Breakpoints and Watchpoints
Description
Type here to search
30°C
ENG
13:18
04-01-2023


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
OnlineGDB beta
online compiler and debugger for c/c++
Welcome, saerahul2003
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Upgrade
Logout
95.3K
GOT AN OPINION?
SHARE AND GET REWARDED.
About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online
input
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit2 10
THE SENTENCE S : abcdehel
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit4
THE SENTENCE S : abcdehellocomputer
Enter choice :(in format [option arguments])
1. Append String to end of S
2. Delete last k charcters to S
3. Print Last K characters of S
4. Undo
5. Exit5
...Program finished with exit code 0
Press ENTER to exit console.
Call Stack
Local Variables
Variable Value
Registers
Register Value
Display Expressions
Breakpoints and Watchpoints
Description
Type here to search
30°C
ENG
13:18
04-01-2023